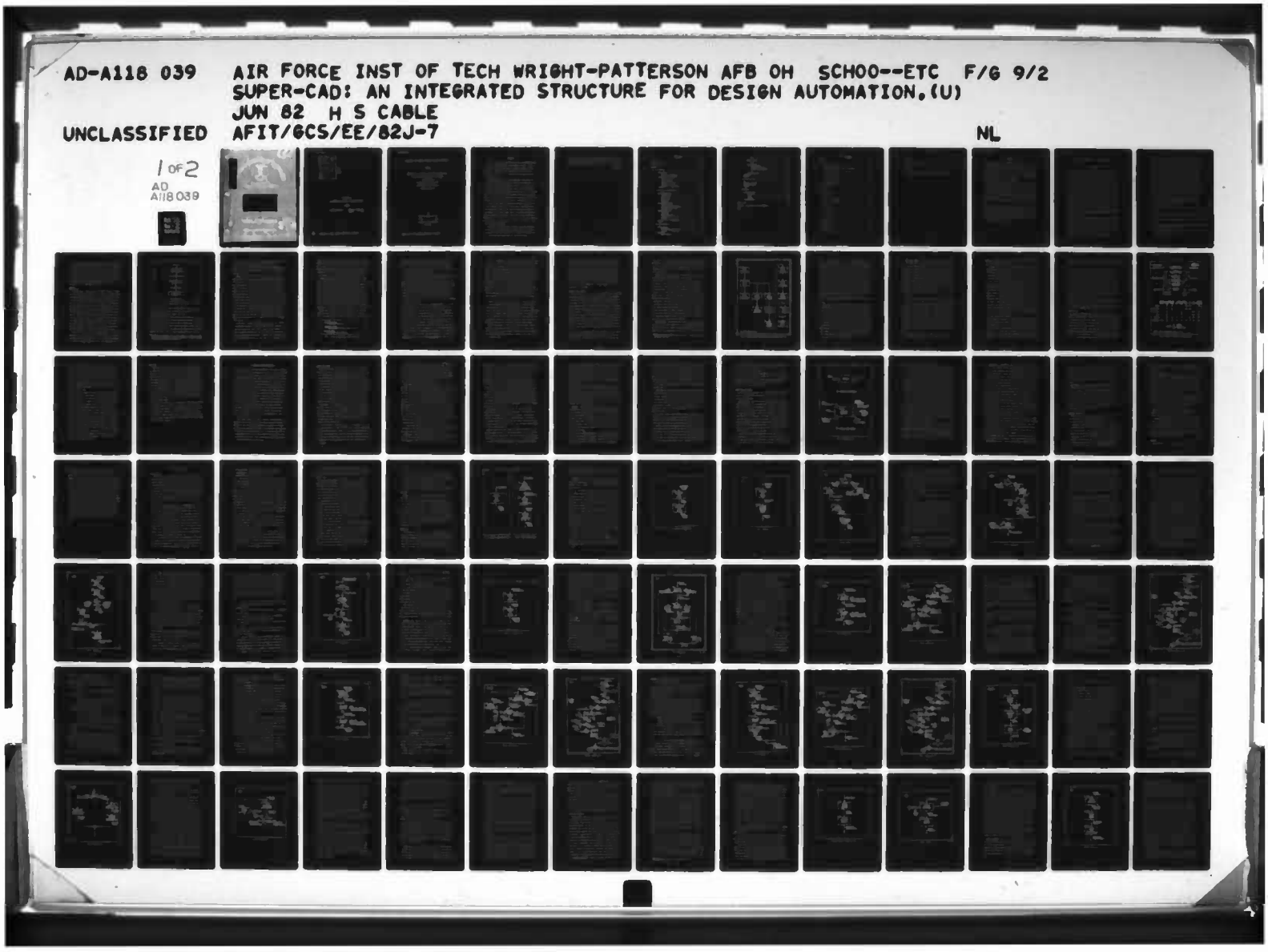
AD-A118 039 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
SUPER-CAD: AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION.(U)
JUN 82 H S CABLE
UNCLASSIFIED AFIT/GCS/EE/82J-7 NL

1 OF 2
AD
A118 039



1 OF 2

AD
A118039

AFIT/GCS/EE/82J-7



# SUPER-CAD:

# AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION

THESIS


AFIT/GCS/EE/82J-7 Hobart S. Cable, II
Major USAF

AFIT/GCS/EE/82J-7

# SUPER-CAD: AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Hobart S. Cable, II, B.S.

Major USAF

Graduate Computer Systems

June 1982

## Preface

This report outlines an approach for developing an integrated system of design tools for designers of complex digital systems. For the purpose of the report the terms Computer-Aided Design (CAD) and Design Automation (DA) are essentially synonomous and can be used interchangeably. I tend, however, to apply CAD more to specific tools and DA to the overall topic.

I became interested in this area through the two Microprocessor Design courses offered by the Air Force Institute of Technology. We used several CAD programs in producing our design, and, while they were very helpful in accomplishing the task, they were not particularly user-friendly. The biggest limitation was that the simulation program inputs (or outputs) could not be interfaced directly with the layout program. In wanting to take advantage of the experience gained in those courses, I decided to work on the concept of an integrated DA system.

I wish to acknowledge the tremendous assistance I received in this effort from Major Al Ross and Major Hal Carter. As my advisor, Maj. Ross inspired my interest in the subject and provided a direction for the research. Maj. Carter became almost a co-advisor in helping refine numerous aspects of the approach. The three of us spent many hours in discussions which led to most of the concepts and approaches presented in this report. I express my deep gratitude to both individuals for their invaluable assistance.

Thanks also to Dr. Gary Lamont, my third thesis committee member, for his efforts to help make this a worthwhile project. In addition, I am indebted to Major Mike Borky who, as an instructor in Digital Computer Design, gave me an appreciation for logic-level design.

I dedicate this work to my family, who endured three and a half years of part-time graduate study on top of my full-time Air Force job. To Charlene and our children, Catherine and Barry, a loving thank you.

Bart Cable

Contents

## List of Figures

## Abstract

This project proposes a structure to integrate a variety of Computer-Aided Design tools into a complete design system. Design aids have provided valuable assistance to designers of integrated circuits over the last decade. However, greatly increased circuit complexity, with the approach of more than a million devices on a single chip, is exceeding the capabilities of current design methods. Greater automation and additional design tools are needed.

A model is proposed which divides the design process into three stages: Specification, Implementation, and Realization. The Implementation stage is examined in detail. Design requirements can be described at different levels of abstraction, from a description of overall behavior to specific gate-level logic details. The model attempts to satisfy the requirements at the higher levels using existing implementations before resorting to the design of new circuits at the lowest level.

The proposed system is an integral part of the Air Force Institute of Technology's increased emphasis on Design Automation. As future efforts address more of the details and problem areas, design tools will be developed to support it. The system will be highly flexible, based on a great degree of interaction with the user, and adaptable to changing technologies and requirements.

# SUPER-CAD: AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION

## I. Introduction

In the last few years, developments in microelectronics technology have accelerated rapidly, and the era of VLSI and VHSIC (Very Large Scale and Very High Speed Integrated Circuits) has begun. These advanced chips have the potential to contain "hundreds of thousands of transistors" (Refs 1:34 and 68). As Integrated Circuit (IC) chips have increased in complexity, the difficulties in design have also increased for IC designers. As a result, the field of Computer-Aided Design (CAD) has evolved to assist the designer with many of the "routine and often mundane" tasks such as "bookkeeping and consistency checking" in keeping track of the basic elements of the design (Ref 2:48). With such help, designers have been able to keep up with the rapid advances in IC technology. Yet, much of the design process is still a manual operation with the designer creating the specific circuit design himself at the logic gate level. He then uses CAD tools to do automated testing and simulations with his circuitry and produce final layouts for IC and circuit board fabrication.

In the VLSI era, as designs are becoming very complex, current CAD tools are no longer adequate. In fact, some observers feel that design tools could become the constraining factor that limits the growth of IC complexity (Refs 3:3 and 72:94-95). Thus, Computer-Aided Design has become a "critical technology" that requires significant enhancement to meet the needs of VLSI in the 80's (Ref 4:3).

Many research efforts are underway to provide that enhancement.

This report summarizes much of the recent work and offers an approach to join old and new CAD tools into a single design system. This "Super-CAD" would allow a designer to input his requirements and work interactively with the system to produce a finished design ready for fabrication. Definition of the system has two aims. First, the system should integrate all steps of the design process into a single package. Second, it should emphasize designs that use families of existing IC chips. While the Super-CAD system will be capable of designing new, customized circuits, it will attempt to solve the problem with available IC's first. This is becoming increasingly important as designs become more sophisticated. With VLSI promising tremendous flexibility and power, existing VLSI circuits may, in many cases, produce more economical designs than would the creation of whole new circuits.

This report will approach the subject in the following manner:

1. Present a general view of the problem, meant partially as a tutorial on Computer-Aided Design and Design Automation to establish a backgound for the research effort.

2. Demonstrate the need for an integrated system to support automated design.

3. Define a framework for integrating design tools into a single structure for digital design in the VLSI era.

4. Propose a model for developing an important part of the structure.

5. Show its relationship to current and future work of the Air Force Institute of Technology and the Air Force.

6. Identify those areas that either require further development or could benefit from future research.

The supporting references provide a compilation of much of the work that has been done on Design Automation over the last ten years. While this report only touches lightly on many of the sources, the bibliography is quite extensive and provides an excellent summary of current activity. In turn, the references listed within these sources can provide further information. In addition to the references specifically cited in the report, the bibliography contains others that also relate to Design Automation and Computer-Aided Design.

## Background

The designers of digital circuitry go through a number of phases when they transform the statement of a problem into a completed design. One model of this process might divide it into three distinct stages. In the first stage, the designers analyze the problem to define the requirements and arrive at a set of specifications. During the next stage they might attempt to meet those specifications with currently available circuits. If some or all of the requirements cannot be met, they must design new circuits. Then they run simulations to test their designs. Finally, in the third stage, they develop a layout of their circuitry and produce a mask set for fabrication of the IC's and/or circuit boards. These three stages of development can be called Specification, Implementation, and Realization (Ref 13:1322). (See Figure 1.) Design Automation (DA) (including the tools of Computer-Aided Design) assists the designer at various points throughout these stages.

DA can be defined as "the art of utilizing digital computers to help generate, check, and record the data and the documents that constitute the design of a digital system" (Ref 5:2). It relieves the designers of repetitive manual tasks, reducing time and costs, and allowing them to

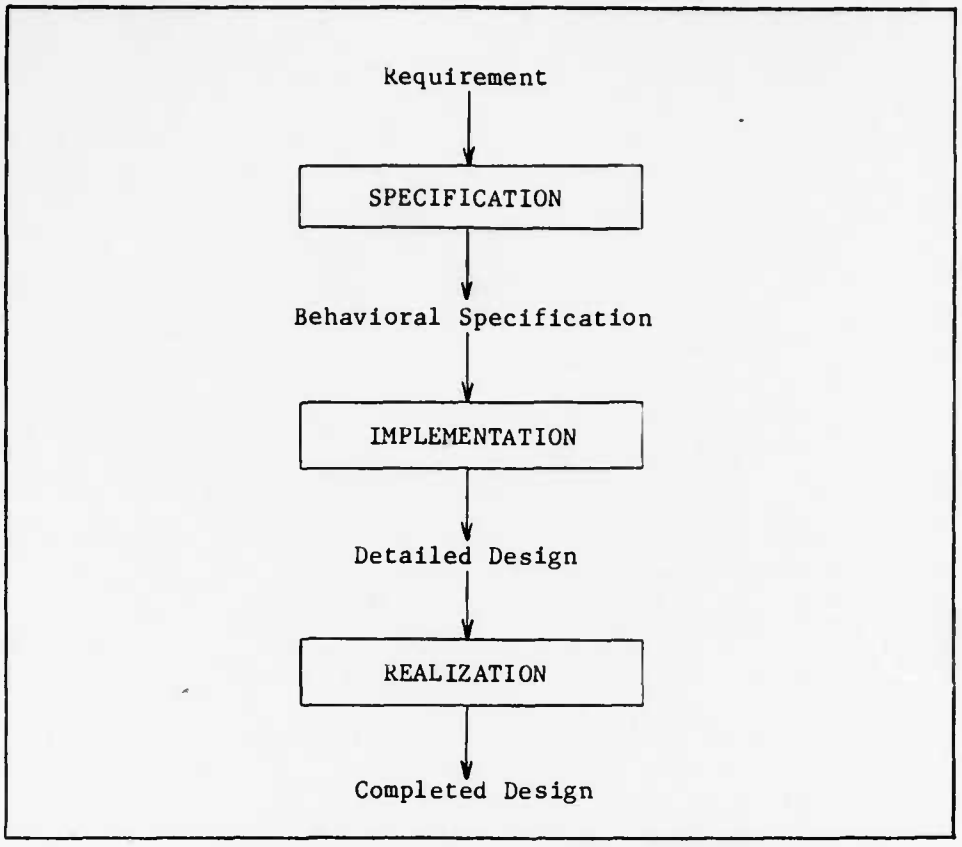



Fig 1. Design Stages (Ref 13:1322)

devote more attention to design problems (Refs 5:2 and 6:100). A study has shown that, for "aspects of the design that have been automated, design time is significantly reduced" (Ref 51:67).

As integrated circuit technology has evolved, development of design aids supporting DA has been largely uncoordinated, with specific systems produced to address specific problem areas or technologies. As a result, a myriad of CAD systems have been developed in recent years (Ref 7:1189). But we are now on the verge of "VLSI systems of enormous functional power" (Ref 8:137)*. With the potential for over a million

*This referenced book by Mead and Conway appears to be the definitive text on VLSI electronics. Virtually every entry in the bibliography from the last three years has cited it.)

devices on a single chip (Ref 9:617), it has become increasingly important to "replace the quagmire of standalone [CAD] applications" (Ref 10:126) with integrated DA systems "oriented around a common data base" (Ref 14:550) and "dedicated to the problems of IC designers in the VLSI era" (Ref 11:552).

History. (Refs 7:1189-1190,1197,1198 and 6:89,100) The history of design automation shows a gradual development of design tools starting in the late 1950's. "Typical applications were automated logic diagramming, [computer] back-panel wiring with discrete wires, and electrical load checking" in 1958 and 1959 (Ref 6:89,100). The 1960's brought printed circuit boards (PCB's) and the use of a large number of solid state components. Design aids at that time focused mainly on circuit analysis and simulation techniques for discrete circuits. Examples are NET-1--1964, CIRCUS--1967, and TRAC--1969. Then integrated circuits arrived, and by the early 70's the use of computers became indispensible as record-keeping tools and means of verifying the design before a chip was manufactured. IC simulators of that time included BIAS-3--1970; CANCER, SLIC, and TIME--1971; ASTAP--1973; and SPICE2--1975. By the middle 70's tools were developed to automatically generate the physical layout and interconnections of the components within an IC, for example, CRITIC in 1974.

At this point it became obvious that "computer-aids were a necessity in the design of complex IC's, both for physical and for functional design and verification" (Ref 7:1190). Gradually, whole systems of loosely-coupled CAD programs have come into being, and CAD techniques have also been applied to the design of complex PCB's. But these programs are largely incompatible with each other. They frequently use a

variety of data formats and often require "manual intervention [by the designer] to move from one program to another." The only true integrated DA systems currently in existence are for some "highly specialized design approaches" (Ref 7:1189-1190). Now that the use of complex and sophisticated IC's is becoming more widespread, the development of a truly integrated system of design tools is becoming a necessity.

Levels of Abstraction. As electronic circuitry evolved into more complex designs, different ways to represent those designs produced hierarchical levels of abstraction (Ref 7:1190). While a number of different levels exist, most of them can be arranged in three general levels. Different authors describe these levels of representation somewhat differently (Refs 7, 12, 13). The Thomas approach was chosen for the project because it provides the best overall definition that is consistent with the Super-CAD research (Ref 12:1201). Note that these levels are distinct and separate from the three design stages. The levels of abstraction are ways to look at a design primarily within the Implementation stage while the design is being created, as shown below.

Thomas describes the three levels as Behavioral, Functional, and Logical (or Physical) (Ref 12:1201). (Figure 2.) Each level of abstraction deals only with information applicable to that level; it is

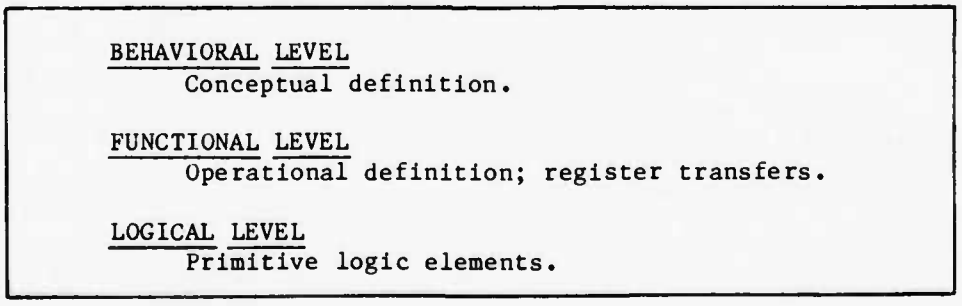



Fig 2. Levels of Abstraction (Ref 12:1201-1202)

not concerned with lower levels of detail (Ref 13:1323). The highest Behavior Level provides a "clear conceptual" definition of a digital design. This level does not consider the structure of how system functions are implemented. Rather, it specifies the overall behavior of the system (e.g., inputs and outputs). (Ref 12:1202)

The Functional Level, on the other hand, begins to deal with some of the specifics of the design. It generally defines operations or functions to be performed (Ref 12:1202) at the register-transfer level. It provides a framework for interpretation of "a sequence of instructions" to cause "action upon a data structure" (Ref 13:1324). Details "of the actual implementation" are still hidden, however. Those are left to the Logical level, which "specifies the interconnection of primitive logic elements." The primitive elements can be logic components, like gates or flip-flops, or larger modules composed of the basic logic components. (Refs 12:1201 and 13:1323)

Thomas provides an example of the differences between levels. Suppose the digital system is to perform the operation "A=B+C". The Behavioral level only views it as "A:=B+C" (":=" means "replaced by") where a new value A is generated from values of B and C. The Functional level may see it as requiring three registers and an adder. The adder takes the values of B and C from the first two registers and places their sum in the third register. The Logical level of the design may actually include a large number of registers and more than one adder, so that the operation could be performed by any one of several combinations of these components. The "boundary" between the Functional and Logical levels may at times be "fuzzy", but the distinction is still between specifying an operation and actually implementing it. (Ref 12:1202-1203)

As a further example, consider a requirement to design the digital circuitry for a simple, four-function calculator. From the Behavioral standpoint the requirement can be described as needing to support the four basic operations of addition, subtraction, multiplication, and division, plus a few extra parameters such as the number of digits and whether chain operations are to be included. The four operations can be expressed in the same "A:=B+C" form as above for addition. A Functional level description, on the other hand, would define the necessary registers and functional blocks to perform the operations. One block could represent each operation: adder, subtractor, multiplier, and divider. Also included at this level would be how the chain operations are to be performed; for instance, each operation might have to be completed (pressing the "=" button) before the next one can be joined to it.

Finally, at the Logical level details of the actual implementation (or possible implementations) are defined. It is here that the different possibilities for accomplishing the four operations are expressed. For example, the functional blocks could actually be implemented with true hardware modules, such as a multiplier to do multiplication. Or, especially for a simple calculator, all four operations could be handled by some form of addition (see Ref 130, Chap. 3) so that only an adder and the necessary registers and counters might be used.

Siewiorek describes a "circuit level" below the logical, made up of transisters, capacitors, and the like (Ref 13:1323). But, as we shall see, many CAD tools currently support the logical level so the designer need not be concerned with the specific elements at the circuit level (Ref 25: Part I, 4).

The design level hierarchy just described provides a mechanism to help simplify the task of the designers. By being able to express the requirements of the design at a higher level of abstraction, they can leave management of the details to the computer (Refs 7:1197; 16:85; and 17:67). This again saves time and money in the design process (Ref 15:314). Research indicates that such high-level representation can be successful. Studies at Carnegie-Mellon University (CMU) have shown "that by increasing the level of abstraction of the basic building blocks of the design hierarchy . . . , a much improved [IC layout] can be produced" (Ref 16:89).

Evolution of the Model

Up to this point, in the evolution of integrated circuit technology and design automation tools, two representations have been described. On the one hand, stages of digital design divide the design process into three areas. On the other, the three levels of abstraction provide different degrees of detail concerning design requirements and implementations. The previous example of an electronic calculator illustrated the levels of abstraction. Extending that example to the design stages, the Specification stage takes the requirements and parameters for the calculator and turns them into a set of specifications. The Implementation stage produces a design from those specifications, operating at one or more of the levels of abstraction. Then, in Realization the design is used to create a layout for the actual digital hardware for the calculator.

Thus, the two representations are not equivalent. They do, however, produce some interesting patterns in the evolution of DA tools and models. First, they highlight a gradual shift from "design of" new

IC's, which relies on basic gate-level building blocks, to "design with" existing IC's, where the IC's themselves are the building blocks. They also show an increasing trend toward integrated DA systems. Several examples represent steps in this evolution and demonstrate how it has led to Super-CAD.

CLODS. In the early 1970's several projects here at the Air Force Institute of Technology (AFIT) went together to support CLODS, the Computerized Logic-Oriented Design System. (See Refs 18, 19, 20, 21, 22, 23, and 24.) CLODS viewed digital design as a six-step process: (1) "reduction of a higher-order language" (DDL*) to "state tables and Boolean equations", (2) "state table minimization", (3) "state assignment", (4) "logic hardware realization", (5) "logic diagram documentation", and (6) "logic circuit simulation" (Ref 24:1-4). Separate programs had been developed for each of these steps, and CLODS drew them together into a compatible system. The system used libraries in secondary memory to store these programs so that only the "currently executing" modules inhabited main memory (Ref 24:16). An additional project produced an extension to CLODS to automate the "design of integrated circuit masks" (Ref 19:ii). Thus CLODS dealt with the Implementation and Realization stages of design at the Logical and Functional levels of abstraction and integrated several CAD tools.

Design Courses. The Microprocessor Design courses here at AFIT currently use the DEC-10 computer of the Avionics Laboratory and several CAD programs (Ref 25). Figure 3 shows the different phases. Students work up their designs manually at the logic gate level. They then enter

*Digital Design Language--a register-transfer language developed by Dietmeyer (Ref 66).

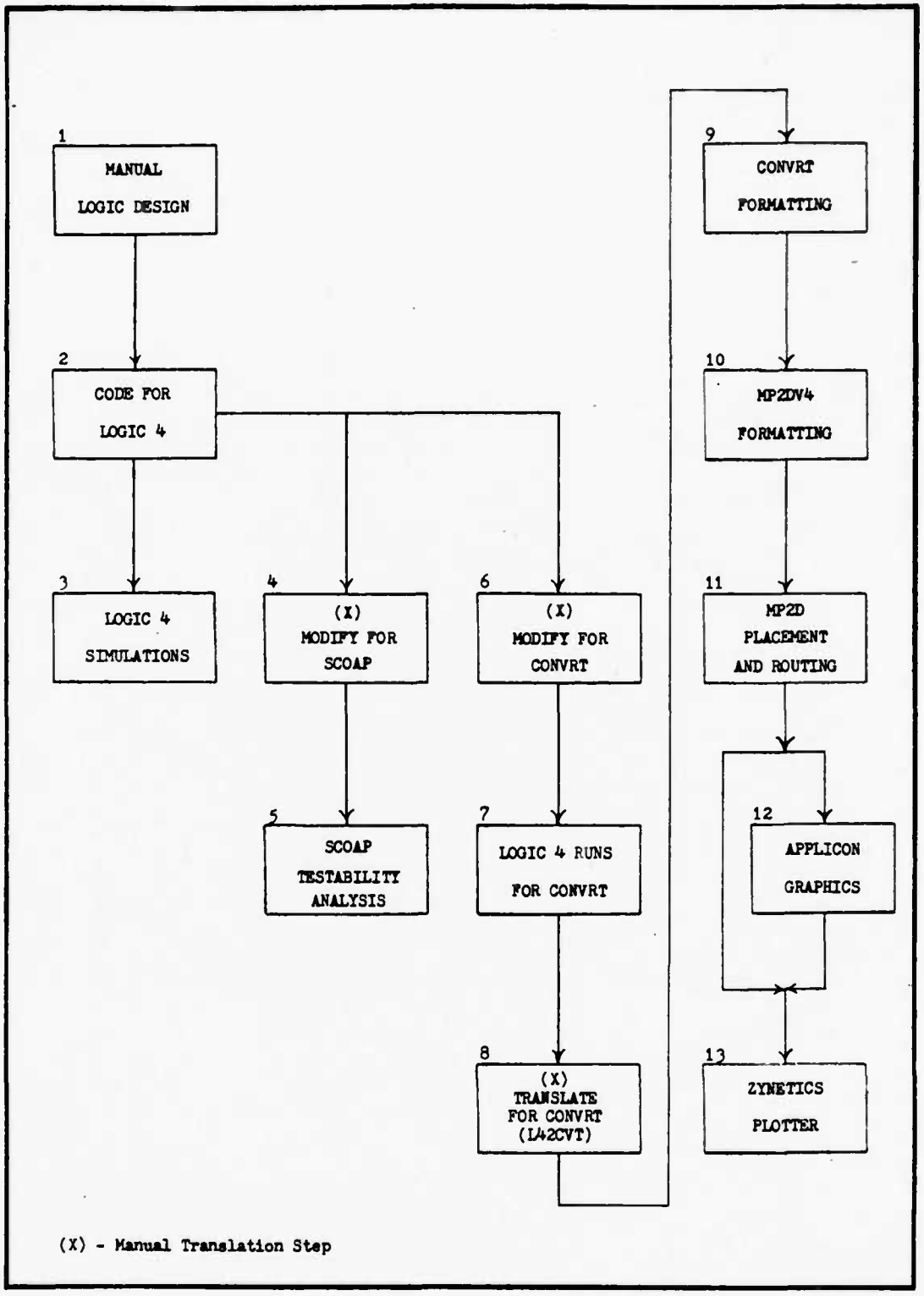



FIG 3. Microprocessor Design Sequence of Operations (Ref 25)

them into a program called Logic 4, which runs simulations to test continuity and timing (Ref 26). The Logic 4 representation can also be used, with minor modification, by a SCOAP program to "efficiently and automatically characterize the testability" of the design (Refs 29:3 and 30). Once a design is finalized, it is run through CONVRT and MP2DV4 programs to format it for the layout program (Ref 27). The MP2D program accomplishes placement of the basic IC components and routing of the connections between them (Ref 28). The result is a graphical representation which can be interfaced to an Applicon interactive-graphics terminal and/or sent to a Zynetics Flatbed Plotter. The final layout can also be used to develop a set of masks for IC fabrication. (Ref 25) This whole process, then, operates within the Implementation and Realization stages and uses CAD building blocks at the Logical level. (Refer to Chapter II, Translation Between Blocks, for additional discussion.)

Univ. of Conn. A project at the University of Connecticut has replaced the standard gate-level building block library with a library made up of the Intel 3000 family of bit-slice microprocessors. The object is to allow "the designer to operate at a high level of [abstraction] in the design process." Thus he can express his design needs at the Functional level and leave the "device details" to be "handled by the automated design system". This system operates essentially in the Implementation design stage, although the users do slip into the Realization stage when they develop a hardware prototype of their designs for further testing. (Ref 31:20-22) A limitation of the system is that it can only produce designs that use combinations of the 3000 series chips, but it is a step in the right direction.

Lawrence Livermore. A further step is the work of Matelan and Ross for the Lawrence Livermore Laboratory (LLL). (Refs 32, 50, 33, 34, and 35) Matelan considered the problem of automating more of the design functions in developing dedicated real-time control systems. He proposed a unique Control System Design Language (CSDL) by which a designer could specify a problem. He sought to simplify the problem specification step and automate hardware selection and software production. An important element was to remove all hardware considerations from the problem definition, allowing hardware components to be selected at different stages as the design evolves. At the same time the necessary software can be developed, aiming for as much concurrent hardware/software development as possible. (Refs 32:1-19; 50:462-463; and 33:25)

Ross extended Matelan's work by focusing on the automation of hardware selection. He proposed a systematic automated design process. Once the designer generates a problem statement in a high level programming language, the computer goes through several development stages to approach the problem. The key element is a library of pre-designed realizations and supporting technologies. The problem statement is specified, tested, and translated into an intermediate form before the hardware realization technology is considered. Ensuing steps choose and test a microprocessor family and produce the necessary software. The designer thus exercises creativity in specifying the problem statement, and the computer completes the remaining design steps. (Ref 34:1-8 and 35:227)

This work has advanced design automation significantly and is a basis upon which Super-CAD is built. It operates within both

Implementation and Specification stages at the Behavioral and Functional levels, through a highly integrated system.

CMU-DA. The last few years have seen a major undertaking in DA by Carnegie-Mellon University. (Refs 36, 38, 44, 42, 16, 2, 49, 43, 37, 45, 48, and 47) The CMU-DA system is "an approach to translating high level behavioral design descriptions into detailed designs at the logic description level" (Refs 44:479 and 42:1052). The project's goal is "to provide a structured design tool" which is "responsive to changing technology" and helpful to designers in exploring different alternatives for implementing a design (Ref 36:93). Inputs to the system include a behavioral description of the problem in ISP (the Instruction Set Processor hardware description language; see Refs 39, 40, 41, and 16), plus paramenters of the design ("optimization criteria") and a library of the "hardware components available to the design system" (Refs 36:93; 38:73; and 16:86). (Note: What the CMU researchers are calling a "behavioral description" is really a blending of the Behavioral and Functional levels of this project, with many elements from register-transfer theory at the lower level. The "Behavioral level" of the Super-CAD project is meant as a true description of overall behavior, quite separate from register-transfer level considerations.)

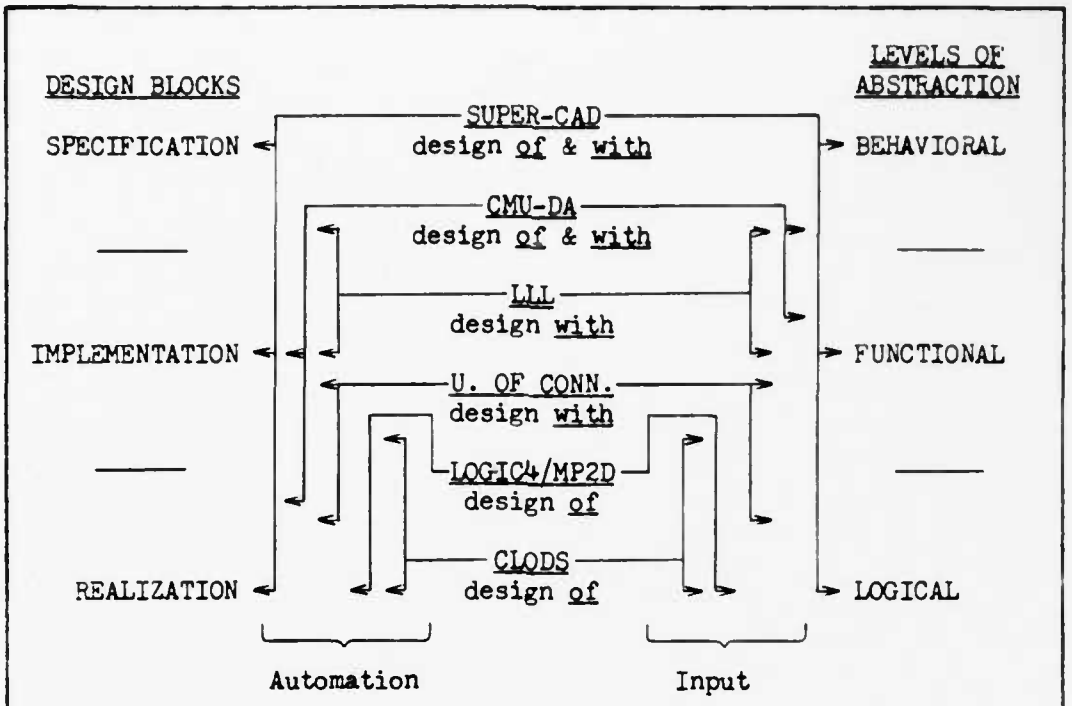
The input of a behavioral specification, while "characterizing the [desired] input/output behavior", does not necessarily dictate the implementation's "internal structure." As the design proceeds through the CMU-DA system, binding implementation decisions are made "in a top-down manner." As each level is traversed, "more and more structural detail is frozen", with the "most influential" design decisions being made first, "to cut down the design search space." The result is a

"complete hardware specification" expressed "in terms of the basic building blocks" used in the design. (Refs 36:93; 38:73; 16:85; 43:635; and 2:50)

The design system is made up of a number of components (see Ref 36), and research is continuing toward completing them all. An important aspect of the system is that different "design styles" may be used (e.g., microprocessors, TTL chips, standard logic cells, etc.) as well as different technologies (e.g., TTL versus CMOS). Flexibility is assured by the module set library, which allows new modules to be added as new VLSI circuits and technologies become available. The system ultimately maps operations derived from the behavioral description to modules from the library based on the design style and technology chosen. (Refs 2:49-50; 36:94; 17:68-69; 44:479; and 7:1197)

The CMU system is another important advancement in integrated design automation. It will automate portions of both the Implementation and Realization stages of design, and is based upon a specification expressed in the ISP language which operates at the Behavioral and Functional levels.

Emergence of Super-CAD. The evolution-through-example just described shows some definite trends. These trends are summarized in Figure 4. (Recall from earlier in the chapter that the three stages and three levels are not equivalent.) The early CLODS work, and current CAD programs (exemplified by Logic 4-to-MP2D), deal strictly in the design of new circuits and/or boards. While they affect automation at both the Implementation and Realization stages of design, they operate solely at the Logical level of abstraction. The University of Connecticut project with the Intel 3000 family, though limited, concentrates on automating




| | Degree of Automation | | | Input Levels | | | | | INTE- |
|---|---|---|---|---|---|---|---|---|---|
| | SPEC. | IMPL. | REAL. | BEH. | FUNC. | LOG. | OF | WITH | GRATION |
| CLODS | | XX | XX | | | XXX | XXX | | XX |
| LOGIC4/MP2D | | XX | XX | | | XXX | XXX | | |
| U. OF CONN. | | XX | X | | XX | X | | XXX | XX |
| LLL | X | XX | | X | XX | | | XXX | XXX |
| CMU-DA | | XX | XX | X | XX | | XX | XX | XXX |
| SUPER-CAD | XX | XX | XX | XX | XX | XX | XX | XX | XXXX |

Scale: X - Slight
XX - Average
XXX - Almost Total

(NOTE: These are example systems only.)

FIG 4. Trends in the Evolution of Super-CAD

the Implementation stage predominately at the Functional level, and seeks a design with existing circuits. The Lawrence Livermore work climbs higher by designing with more than one family of microprocessors and begins to address problems in the Specification stage and the Behavioral level. The emphasis, however, is still on automated Implementation and Functional descriptions. The CMU-DA project, on the other hand, combines Behavioral and Functional level specifications and covers both design of and with. It does not attempt to automate any of the Specification stage.

With the trends viewed in this manner, it is a logical extension to attempt to automate all three stages in the digital design process and to address a problem at all three levels of abstraction. Super-CAD is intended to provide that capability and flexibility. Thus, as shown in Figure 4, Super-CAD will automate as much of the Specification, Implementation, and Realization stages as possible. Automation of portions of the Specification stage will allow the designer to input requirements in a simpler form than a complex hardware description language. Also, Super-CAD will operate at all three levels of abstraction, not just the Behavioral level. It will first attempt to complete a design at the Behavioral level with existing circuits. If unsuccessful, it will try the same thing at the Functional level. If that is also unsuccessful, it will move to the Logical level, attempt to finish the design with available IC's, and produce new circuits for any remaining parts of the specification. Therefore, Super-CAD makes a strong effort to satisfy all or part of the requirements with existing IC's before resorting to the design of new ones.

## Overview

The structure of Super-CAD is based upon the design stages of Figure 1. This report defines the structure in detail, analyzing these major stages in reverse order. Since the Realization stage is currently the most highly automated (Ref 51:60-61), the part of the Super-CAD model which draws the automated tools together is described first, including a few examples of applicable tools--both those in existence now and those that need to be defined. Next, a detailed model is proposed for automating the Implementation stage. This is an extensive part of the project and will include many DA tools that require development. Finally, the Specification block is examined. Since many important decisions are made during this stage, it is the hardest to automate and requires the greatest future efforts.

Chapter II presents the overall philosophy of Super-CAD, explaining the general concepts of what the system should do. Then Chapter III describes the Super-CAD model in the sequence given above. Chapter IV ties Super-CAD in with the work in design automation being conducted here at AFIT. Chapter V concludes the report by summarizing the model and highlighting the many areas where future research is needed to help the model become a reality.

## II. Philosophy Behind Super-CAD

Initial research into design automation for this project revealed that a wide variety of CAD programs are available. Most of them address specific areas in DA and generally are not compatible with one another. In the midst of this diversity, however, is a common basis for assisting the digital designer with routine, repetitive tasks. Their use has supported the development of more complex and sophisticated designs. But with the dawning of the VLSI/VHSIC era, it has become apparent that the development of CAD tools needs to be directed into a more coordinated effort, and integrated systems that automate virtually every phase of digital design should be developed. Super-CAD is meant to focus attention on these needs and provide a framework within which they can be satisfied. It defines an integrated structure that is highly user-friendly and flexible, allowing a designer to input a simplified description of requirements and interact with the system to produce a design. Or, he may choose to use only a portion of the facilities available in Super-CAD; for example, to run simulations on a small circuit already designed.

This chapter presents the overall approach to Super-CAD--some of the thoughts behind arriving at an all-encompassing system for design automation. After many of these thoughts were organized, a pertinent paper by Daniel and Gwyn was discovered which describes work being done at Sandia National Laboratories (Ref 52). It turns out that their approach agrees with many of the areas presented in this chapter, including the use of an overall "Executive" program and supporting a variety of existing IC families and technologies.

General Philosophy

Super-CAD will represent an integrated system of tools, ultimately automating portions of all phases of digital design, from inputting a set of requirements to producing a mask set for hardware fabrication, including the software and/or firmware to operate it. The goal is to allow the designer to input a relatively simple description of the problem without having to break it down manually into many details, such as those required by a complex hardware development language. The eventual Super-CAD configuration should come as close as possible to producing a final design completely automatically, once it receives that simplified input. Even if such a configuration is attained, however, Super-CAD must still be a highly interactive, flexible collection of design tools--a "toolbox"--that aids the user in any chosen phase of design.

Flexibility. It is imperative that the structure of Super-CAD be flexible, as follows. Its actual development will be in stages, with individual modules or blocks being completed relatively independently. The system should allow these tools to be used separately, once they are available. Translation programs may have to be used between tools, although the aim must be for as much standardization as possible so that modules can be interfaced easily.

An early function of Super-CAD will be to draw together many of the CAD tools currently in existence. For instance, the Logic 4/CONVRT/MP2D set of programs described in Chapter I can be incorporated into the system and used separately until interface programs are developed. After such tools are interfaced directly into Super-CAD, they can be modified and expanded to include a greater number of design building blocks and technologies.

The key to Super-CAD's flexibility, then, will be a dynamic approach to developing it in stages, adding new tools as they become available, and replacing or modifying old ones as necessary. Included in this should be the ability to recover from mistakes or unrealizable directions. Since Super-CAD will be an amalgam of many modules and subsystems, when one is found to be unproductive or incorrect, it can be removed and replaced without affecting the whole system.

User-Friendly. Super-CAD must be appealing to users from the very beginning. As development commences on individual tools, these must be straightforward and easy to use. The design community will be very suspicious of such an all-encompassing system as Super-CAD promises to be. But its early stages should help to lessen such suspicion by presenting a system of tools that designers can benefit from. The tools should be in terms the designers are familiar with; Super-CAD should speak their language. As more of the system is filled in, more users should find it helpful. If they are satisfied with what they have been using, they may want to try the new portions. The aim in the short term is to make the system useful for a variety of designers. In the long term, familiarity with the system should allow them to rely on it more and more, so that ultimately it can assume most of the design functions--though still under their control, if they so choose.

So, while Super-CAD will evolve into a high level system automating the majority of the design steps, it should always retain the "toolbox" flavor with the ability to assist designers at any level.

Executive. A feature that will make this possible is the use of an Executive--a master program to control all the toolbox programs and, most importantly, interface with the user. It is the Executive that

will welcome users onto the system and offer them a list of options (a menu) for use of the system. In the early stages of Super-CAD, the Executive will provide access to the available tools, but the user may have to manually translate from one block to another. As the system develops, the Executive will take over these chores and provide the translation automatically. At first, such translation between tools may be rather unwieldy. But this will be overcome as the system grows more sophisticated. Old tools can be replaced by new programs that do not require translation; they will interface directly with other modules.

An important function for the Executive is to keep the designer informed of how the design is progressing. At any time the designer should be able to find out which Super-CAD module is currently working on the design. This assumes that major portions of the design are being done by the system in an automatic mode. Equally as important, however, would be an interactive mode.

Interactive. In the discussion above it is obvious that, to be effective, Super-CAD must be highly interactive. The designer and the computer must communicate directly with each other. As Waxman says, this is a "key ingredient" to the "design environment" (Ref 53:546). Many design problems will require that the computer draw the user into critical portions of the design process, such as during the development of a set of specifications.

When the problem is first entered, the Executive can help the user properly format the data. It can provide specific "prompts" for what "is required next" (Ref 54:347). When the user needs more information, the Executive should provide a "HELP" file for any necessary assistance (Ref 54:348). As the design progresses, the interactive environment

will allow the designer to help the computer solve specific design problems, for example, timing considerations. This may force the designer to think through the problem further and possibly modify some of the input parameters. Or, when the user is monitoring the output of an automatic part of the design operations and detects an error, he can "stop the computation" and attempt to correct the problem before processing continues (Ref 55:1356).

With this highly interactive environment and the many functions included in Super-CAD, sophisticated graphics terminals will be a necessity. Besides providing for various kinds of data entry (Ref 56:109), they will support interactive design modification and layout checking in the Realization stage. Examples of such systems are APPLICON and CALMA (Ref 57:1285). Also, a Graphics Work Station is being designed at AFIT (Ref 58), and it will support Super-CAD.

Database. Another feature that is essential to the sophistication and flexibility of Super-CAD is an effective database system. It is a crucial element in maintaining the vast libraries of design building blocks, from families of integrated circuits to basic gate-level elements in various IC technologies. It can contain previous problem solutions which might be useful in future applications. It can also store new building blocks, or "macros", developed by current users for later use, and keep track of current designs in various stages of completion (Ref 6:90-91).

The need for a common database in an integrated DA system like Super-CAD has been widely proclaimed. (See Refs 59, 54, 14, 60, 61, 62, and 63.) Such a database "integrates, organizes, and controls" (Ref 60:285) great amounts of "design data" (Ref 63:399) throughout the

growth of a design in such a system (Refs 62:394 and 64:104). From the standpoint of Super-CAD users, the "'actual' structure" of the database is not important as long as the "'implied' structure" is easy to use (Ref 54:337). Also, the flexibility of Super-CAD dictates that the database "allow new elements to be added without disruption . . . as technology evolves" (Ref 6:93). (Comprehensive discussions of design automation databases can be found in Losleben, Ref 54, and Eastman, Ref 59.) A database that can support Super-CAD is currently under design at AFIT (Ref 65).

Conclusion. The general philosophy of Super-CAD promotes a system of great flexibility and usefulness, supported by interactive graphics terminals, a dynamic Executive, and a comprehensive database. It will offer many aids to the digital designer and allow him to use any tool or group of tools interactively or automatically. It can serve as pad-and-pencil for the designer to work up his design at the logical level, or provide a library of possible implementations at functional and behavioral levels.

For example, suppose someone needs an analog-to-digital converter. The ultimate Super-CAD configuration will permit him to input that requirement in one simple statement, such as "A/D Converter." Super-CAD would respond by asking for the desired parameters for the component or by offering a selection of components and their specifications. Through this interactive process the user can select his converter from those available and receive a printout with the interconnections and instructions for use. This operation can thus replace the process of manually looking through all the data books to find the right component. Another case might be where a user inputs a set of requirements in a high level

language and interacts with the system to receive as output a wiring list and selected IC's to do the job, plus the binary code for ROM chips--or even a full PCB design.

As the system is developed in stages and users come to rely on the tools it offers, it will have a greater chance for acceptance at the higher levels as a total design package. Its continued acceptance will be supported by its adaptability to advancing technology and the accessibility of its tools at the lower levels.

## Philosophy Extensions

In preparation for the next chapter's discussion of the Super-CAD model, this section offers some intermediate details as an extension to the general philosophy.

Process Blocks. As will be presented in Chapter III, the Super-CAD model is developed using the top-down design methodology, to provide greater detail with each succeeding step. A representation used in the model is the process block of Figure 5(a). As shown, it can represent the whole Super-CAD system. A set of requirements is the input, and a completed design is the output. It can also be a representation of the individual modules at all levels in the model. Applying the concept of functional decomposition (in the software sense), each block can be expanded internally, providing more detail at the next lower level. Each such process is defined by the data that is available at the input and the data that is desired at the output.

Figure 5(b) gives a more detailed view of the process block. Again, it may represent either the highest level of the system or any one of the lower level blocks. In some cases at the lower levels, parts of the diagram may not apply or might have to be modified; e.g., placement-

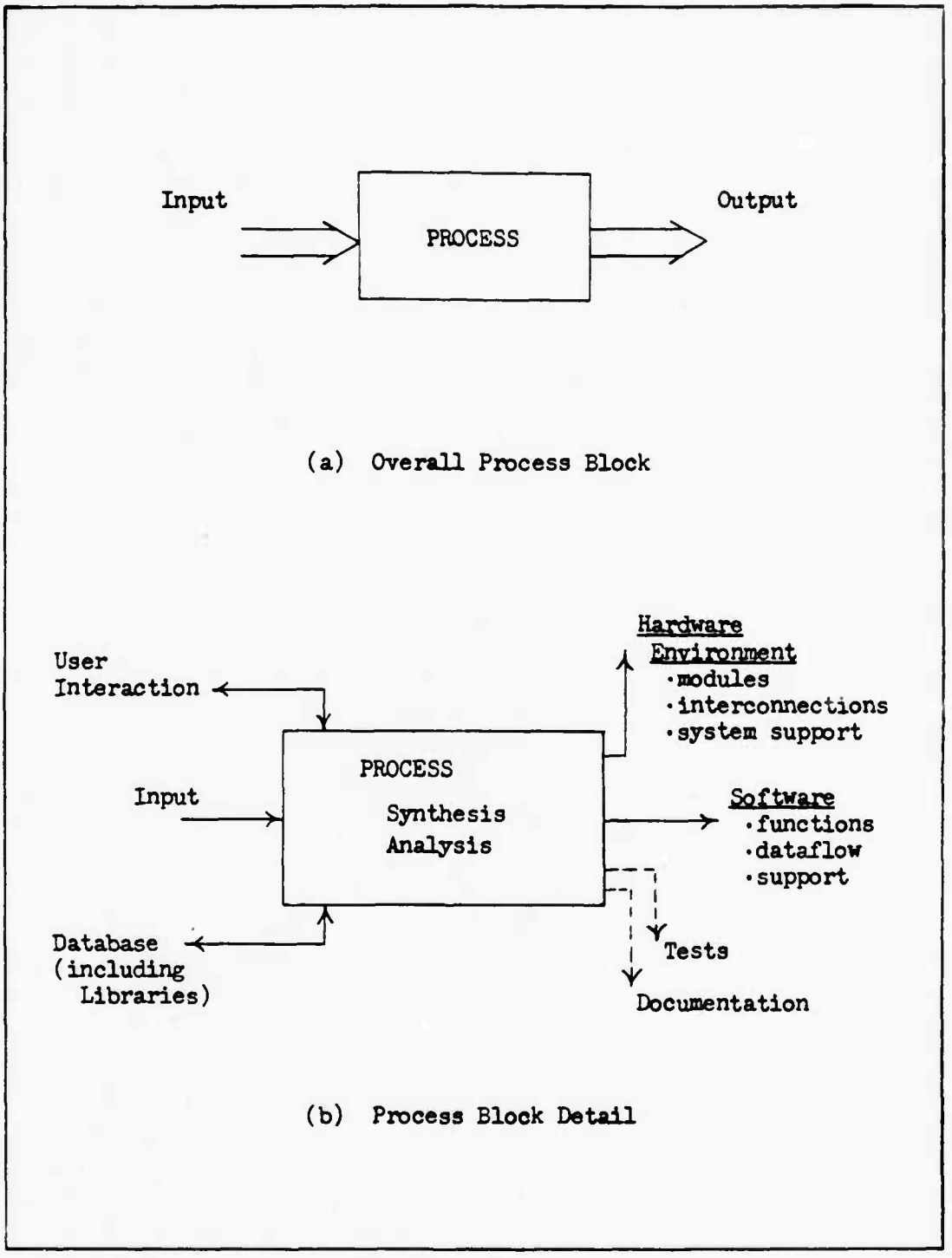



FIG 5. Process Block

and-routing processes apply only to hardware.

That example highlights two important points. First of all, the process block really represents design automation tools or sets of tools. This is true at all levels of the model. At the top, Super-CAD itself is a tool that produces a completed design. Near the bottom, the placement-and-routing process is a tool that produces a layout for an integrated circuit or printed circuit board. Secondly, a process block may contain more than one module at the next lower level. For instance, at certain points in the model hardware and software outputs will split. They may conduct some functions in parallel (such as simulation and testing) but others independently (e.g., placement-and-routing).

Referring again to Figure 5(b), and assuming it applies to any module in the model, the "Input" arrow depends upon what the block is interfaced to, such as the output from a previous block. The "Database" is as described earlier in this chapter, with libraries of existing circuits, previous designs, and current designs-in-progress. "User Interaction" was also explained previously. The outputs on the right may be absent, or present in various forms, as you proceed through the model. In many cases they will be lumped together in intermediate stages of the design before they are broken out separately.

As mentioned earlier, the inside of the process block depends upon the available inputs and desired outputs. In many instances the process may be thought of as "Synthesis" and/or "Analysis". The synthesis portion creates or produces, and the analysis portion tests or examines. Thus, in the Super-CAD model, a process block could merely be a decision point where an input is tested to determine which module to activate next.

Translation Between Blocks. (Refs 25, 26, 27, and 28) Returning to the earlier discussion of how Super-CAD will initially incorporate such tools as Logic 4 and MP2D, this process should be examined in more detail. (See Figure 3.) Actual use of these CAD tools requires many translations to get from an initial gate-level logic design to a final IC layout. First of all, the input to Logic 4 to run circuit simulations will not produce an output that can be used by the next program, CONVRT. Instead, a modified input has to be run through Logic 4, which results in an output full of "warning" messages. That output is then run through a locally-produced program, L42CVT, to strip away the warnings and format the whole thing for CONVRT. The alternative is to manually create a new file of the circuitry in the proper CONVRT format--more straightforward perhaps, but much more work. Even then the design still must go through two formatting programs, CONVRT and MP2DV4, before it can be accepted by MP2D for layout. Thus, only five of the 13 blocks are actual productive processes: 1, 3, 5, 11, and 13. The rest are either input processes, manual translations, or formatting programs--each requiring direct action from the user.

Super-CAD will help designers to overcome such problems. In this instance it would provide automatic translation. If the user inputs a logic design, Super-CAD can format it for simulation and, upon successful completion, format it for CONVRT--all automatically. This is but one example of how the Super-CAD system will remove more of the drudgery currently facing digital designers.

Translating User Input. An extension to the above discussion involves different input languages. While Super-CAD should ultimately be best addressed through an easy-to-use, high level language, it can serve

many more users by accepting languages they are already comfortable with. Much digital design work is being done with Register-Transfer Languages (RTL's) (Ref 66) (e.g., ISP used in the CMU-DA project). If a user wishes to enter design requirements in an RTL, the Executive can call a conversion program to translate that input for the applicable module.

Testability. Digital design using VLSI circuits should consider the issue of testability early in the design process. The time and costs involved require that most chips work properly the first time. The Super-CAD system, by emphasizing designs with available circuits, must assure that testability is included when different modules and circuits are interfaced together. Self-test capabilities should be made a part of the design as much as possible. Throughout the development of Super-CAD, then, testability should be strongly considered early in the design phases and at all levels of abstraction.

## Important Questions

The above discussions on the general philosophy and some of the details involved in Super-CAD lead to questions that should be considered as the model is examined in the next chapter.

The Model. A number of steps are required between entering design requirements and producing a completed design based on a combination of new and existing chips. What are these steps, and what tools and techniques can help automate the ones done manually in the past?

Languages. An important consideration is the use of languages in the system. Possibilities range from using a high order language (HOL) at as many levels as possible, to describing design problems in a hardware description language (HDL) or a register-transfer language

(RTL). Is one language enough? Perhaps the system can start with a structured HDL, such as ISP, and evolve to simpler languages and higher level descriptions.

Hardware-Software. Software development must be considered, and the point at which software is separated from hardware can be important. Where should this point be, and what problems can arise when it is a fixed point? With the integration of hardware and software processes in this model, some software tools and techniques might be adaptable to hardware development. Also, Software Engineering techniques could provide effective tools for the system.

Process Blocks and Tools. In the expansion of the model, can the process blocks be decomposed in alternate ways? Since each block may contain a set of design tools, these should be able to work together and also interface with tools in other blocks. Current CAD tools can be integrated into the system along with newly defined ones.

Database. Can the system be database-dependent rather than language-dependent, translating whatever language the designer wishes to use into the form needed by the database system?

Artificial Intelligence. Artificial intelligence techniques will become increasingly important in design automation (Ref 67). Can they be used effectively in implementing one or more tools for Super-CAD?

These are just a few of the important areas to be examined in the development of the Super-CAD model. Most are addressed in Chapter III. Those that are not, plus additional questions that should be considered in future work, are discussed in Chapter V.

## Summary

Before proceeding to the actual model in Chapter III, it is

important to sum up the Super-CAD philosophy as follows:

A project of the size of Super-CAD is larger than a single Master's Thesis can hope to define adequately. Thus, this report adopts a somewhat lofty, philosophical approach to attempt to grasp the "big picture" of how it all should go together. It could not possibly include all the recent work that has been done in CAD/DA. At times, important points may be oversimplified or obvious points may be overemphasized.

But the overall goal is a flexible and adaptable system. This initial definition must also be flexible/adaptable, and it will become more detailed and specific as work is done by others to support it. This extends to many of the decisions made in developing the model; details are presented as suggested ways of implementing the system, not as the only--or even the best--answers. Besides offering an overall structure to get things started, the aim is to inspire further thinking and research. If portions are found to be oversimplified or erroneous, perhaps such discoveries will motivate work to be done to correct the deficiencies, better define the structure, and help fill in parts of the system.

# III. The Super-CAD Model

## Introduction

As indicated in the previous two chapters, the main goal of this project is to define a structure for an integrated design automation system. A model is developed to provide that definition, and this chapter describes the model. Just as top-down and structured design techniques can be applied to the design of digital circuits and systems, they are applied to the development of the Super-CAD model. Yourdon defines structured design as "the art of designing the components of a system and the interrelationship between those components in the best possible way" (Ref 70:7). That philosophy is employed in the design of the model.

The top-down methodology is also used in presenting the model in this chapter. A systematic approach first gives a general description, then decomposes it into greater levels of detail (Refs 54:329 and 69:618). The design stages (Figure 1) will be examined in reverse order. The Realization stage, where the least new work is required, is analyzed first. Then the Implementation stage is examined in great detail. Breuer says that "logic design requires over 50 percent of the total design effort, yet few automated tools" are available to support it (Ref 51:68). This is the major area of emphasis in the Super-CAD model. Finally, the Specification stage, where the greatest future work will be required, is presented in general terms.

It is important to point out that this presentation of the model is one example of how the system can be implemented and is not meant to rigidly bind follow-on work. Future efforts will determine the actual details of implementation as Super-CAD is developed. Portions of the

model can be examined in greater detail and modified as necessary.

Assumptions. To facilitate the development of the high-level Super-CAD model, several assumptions have been made:

1. When a specification for a problem is partitioned into subsets, the subsets are assumed to be mutually exclusive. In other words, no part of one subset can be part of any other. For example, if a specification included a particular computer instruction set, subsets could be made up of specific instructions that did not overlap. (This does not mean they cannot interact. Some instructions might rely on others, so that a modification to one might impact several.) This assumption simplifies the part of the model where individual subsets are considered for implementation. It assures that a specific, exact implementation of one subset does not affect any others. (The concept of close implementations is examined in section 2.1.1.) Since some problem requirements may not be partitionable into mutually exclusive subsets, the model may not be effective in those cases.

2. The point where hardware and software designs separate from each other is placed arbitrarily at a specific location in the Realization stage. This simplifies the presentation of the model, but also limits its flexibility. Refer to the Realization section, block 3.2.3, for a further descussion of this subject.

3. Thomas describes two major parts of a digital system: control and data (Ref 12:1201). These are not treated separately here, since this model is general in nature and not concerned with the differences. Also, the designer may incorporate his own partitioning of data and control by the nature of his definition of requirements and specifications. Detailed later work should consider them individually.

Each of these assumptions relates to an important area in digital design. Future efforts to carry on the work of this project can enhance it by removing the assumptions and making the model applicable to more general cases.

Notation and Conventions. Presentation of the model relies heavily upon representative diagrams. Several approaches are combined in defining the Super-CAD structure. Principles of structured design are used as much as possible to show, at least at the higher levels, major process blocks with only one input and one output. Although the lower levels stray from this occasionally, the overall result still actively supports structured design. (See Ref 70.) Some of the concepts of Data Flow Diagrams (DFD's) (Ref 71) are incorporated also. The model diagrams combine DFD and flowchart conventions to show data and control together. Data flows are labeled using DeMarco's hyphen convention, and they may include more that one data element. Process blocks use flowchart conventions--rectangles or diamonds--and interaction with the user is shown with circles. Elements of the database are indicated by straight lines, similar to DFD files. (Ref 71:Chap. 5)

A key to the model representation is the use of hierarchical levels. Again following DeMarco, each succeeding level is numbered with an additional decimal point. (For example, 2, 2.1, 2.1.1, etc.) Each separate diagram is a decompostion of only one "parent" process block. When process numbers become unwieldy, only the final digit will be shown on a particular diagram. DeMarco continues leveling until processes can no longer be divided into smaller pieces ("functional primitives"). In many cases, the Super-CAD diagrams do not go all the way to the primitive level, since further decomposition will depend on future research.

(Ref 71:Chap. 7)

The important thing about the diagrams is that they provide a systematic view of the Super-CAD structure. By following the numbering convention, any one process block can be traced down to its lowest level of decomposition. Also, this approach allows us to examine one part of the model at a time, with increasing levels of detail.

The diagrams for the model are presented throughout this chapter and are repeated in the Appendix in a continuous set of figures for easy reference.

Example. In Chapter I the example of an electronic calculator helped demonstrate the levels of abstraction and stages of design used in this project. In this chapter a more complex example is used to illustrate different parts of the model. It is taken from "MADAM": The Micro Ada Machine Project, in which this researcher participated (Ref 25). The aim of the project was to design a microprocessor that used machine instructions corresponding to the pseudo-code generated by an Ada compiler. It was based on an instruction set previously defined by Garlington (Ref 129). The MADAM project succeeded in producing module designs for a major portion of the microprocessor to support a subset of the designated instructions.

## The Model

"Digital system design is, in fact, a sequence of steps and iterations which aim at turning an idea into a physically realized system." Developing "programmed computer aids" for such design efforts requires that "some model [of that] process must be constructed" (Ref 12:1201). Figure 6 shows the overall process block for the Super-CAD model, and Figure 7 is the next level showing the three design stages described in

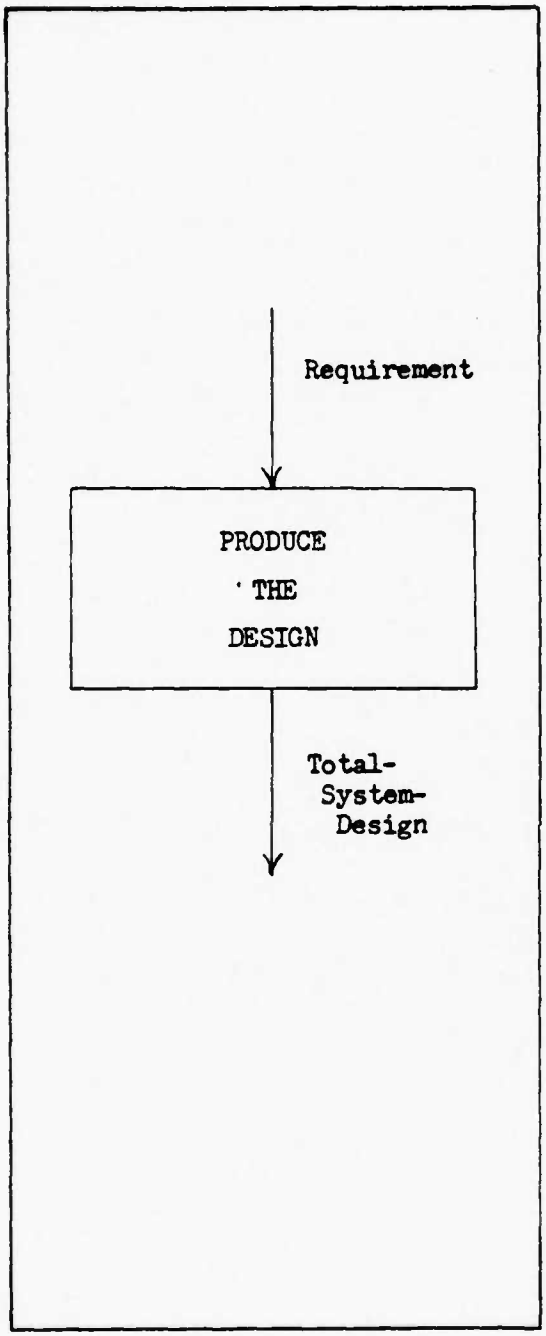



FIG 6. Overall Super-CAD Process

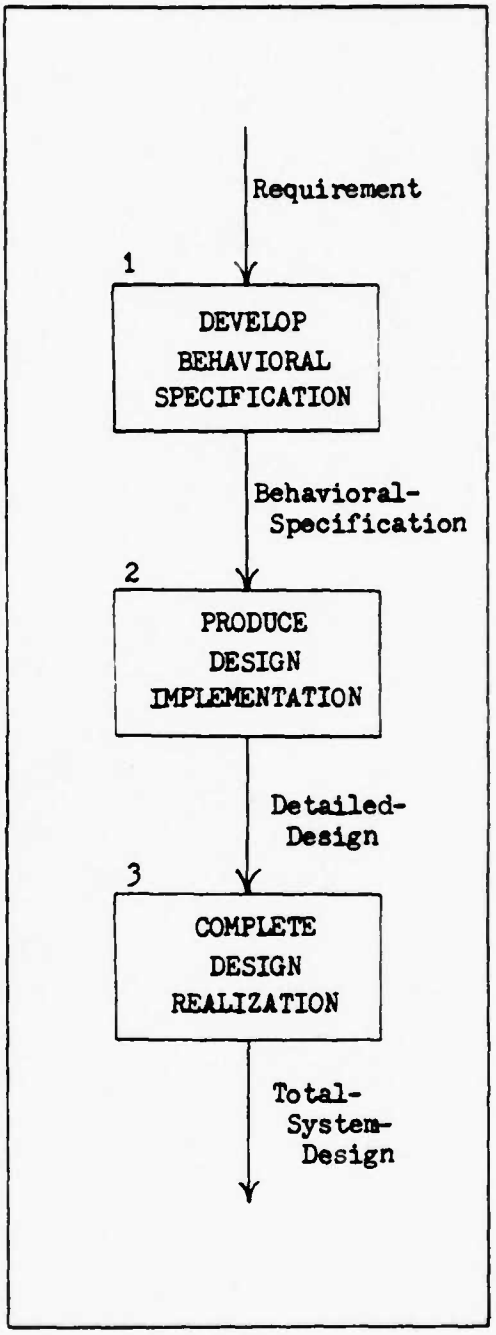



FIG 7. Main Design Stages

Chapter I. Design should start with the input of a requirement at the top. By the time it has progressed through block 2, a detailed design has been generated. Block 3 turns it into a Realization.

Realization. The Realization stage takes the design created in block 2 and produces the actual physical design, with associated software, ready for fabrication. Figure 8 indicates two main divisions within block 3.

3.1. The first division (Figure 9) partitions (3.1.1) the detailed design along the boundaries of the separate IC's defined in the Implementation stage, and then separates hardware and software (3.1.2). (The Implementation stage, block 2 in Figure 7, is described later in the chapter.) The tools to accomplish these tasks are to be defined as the Super-CAD structure is filled in. In this case, they will be straightforward programs that recognize the divisions among the different IC's and between hardware and software already delineated during Implementation. They will assure that the data is structured to be split up by later process blocks. For example, the MADAM project designed separate modules for a number of functions. Many were combined to run simulations and test interfaces during Implementation. If the modules represented different IC's, block 3.1.1 would prepare the data so the boundaries between IC's were clear and the IC's could be separated in later steps. (Note in Figure 9 the use of the symbol "_/" to indicate a process block number that has no lower level diagram.)

3.2. The second division within block 3 is where most of the Realization work is done: layout of the IC's and/or PCB's and coding of the software. Figure 10 gives the expanded view of the constituent blocks.

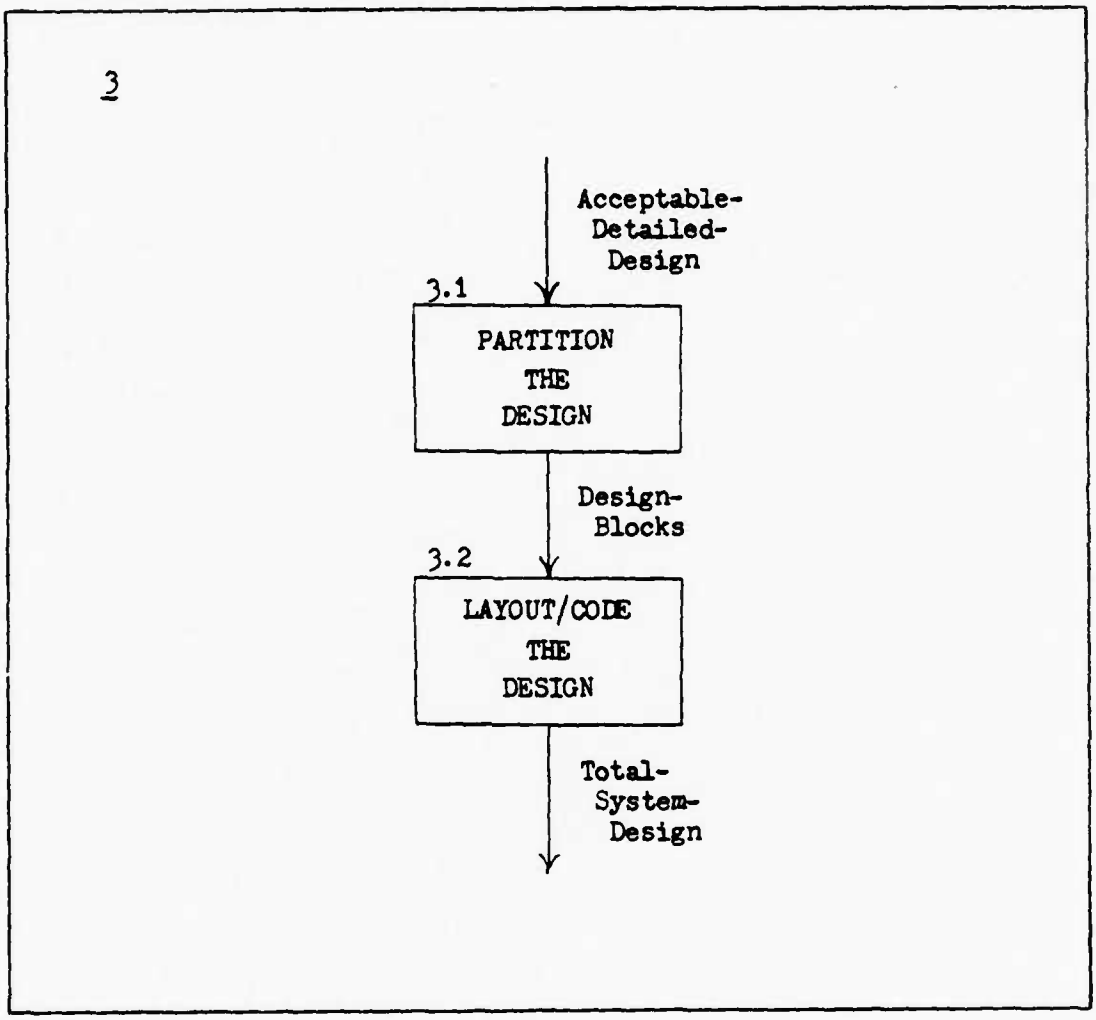



FIG 8. Block 3 - Realization

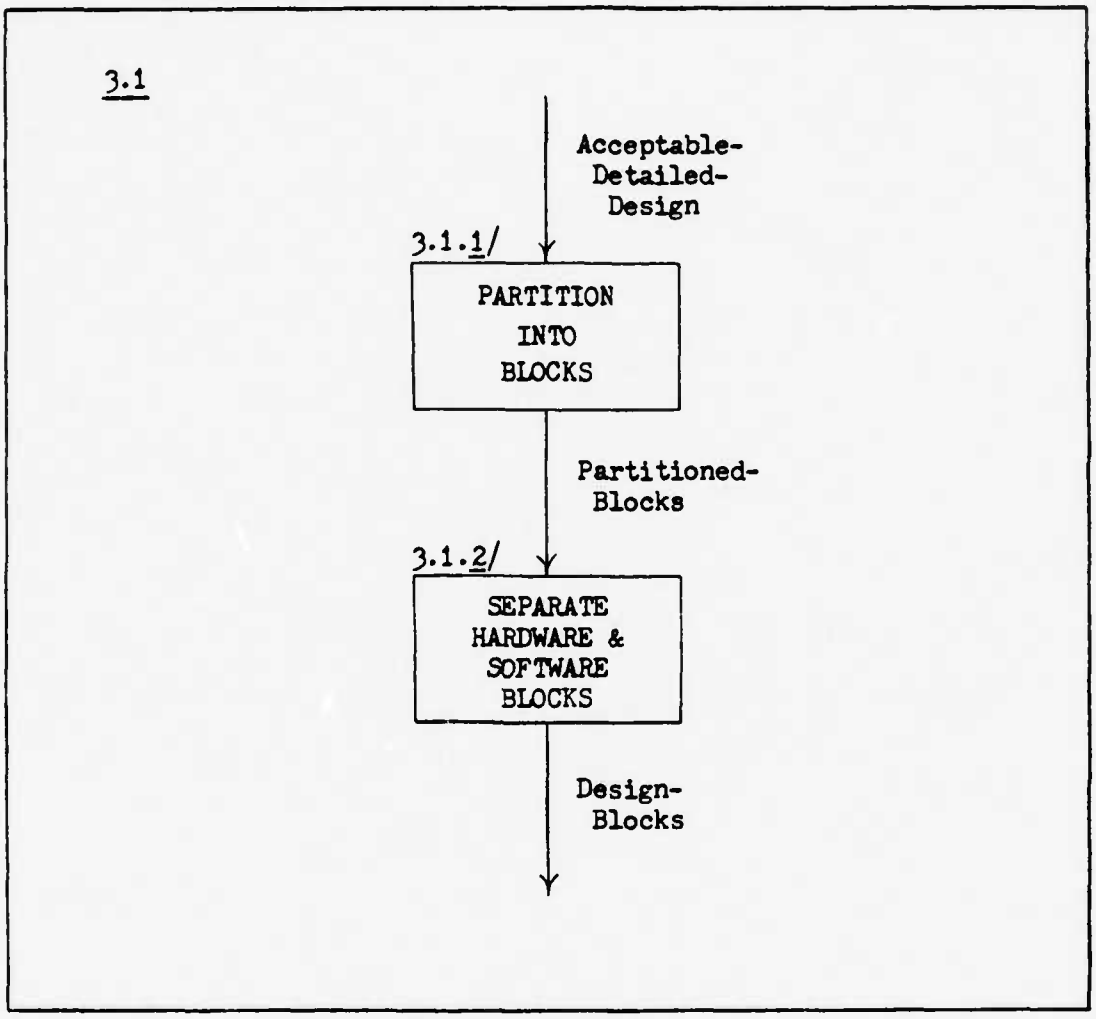



FIG 9. Block 3.1

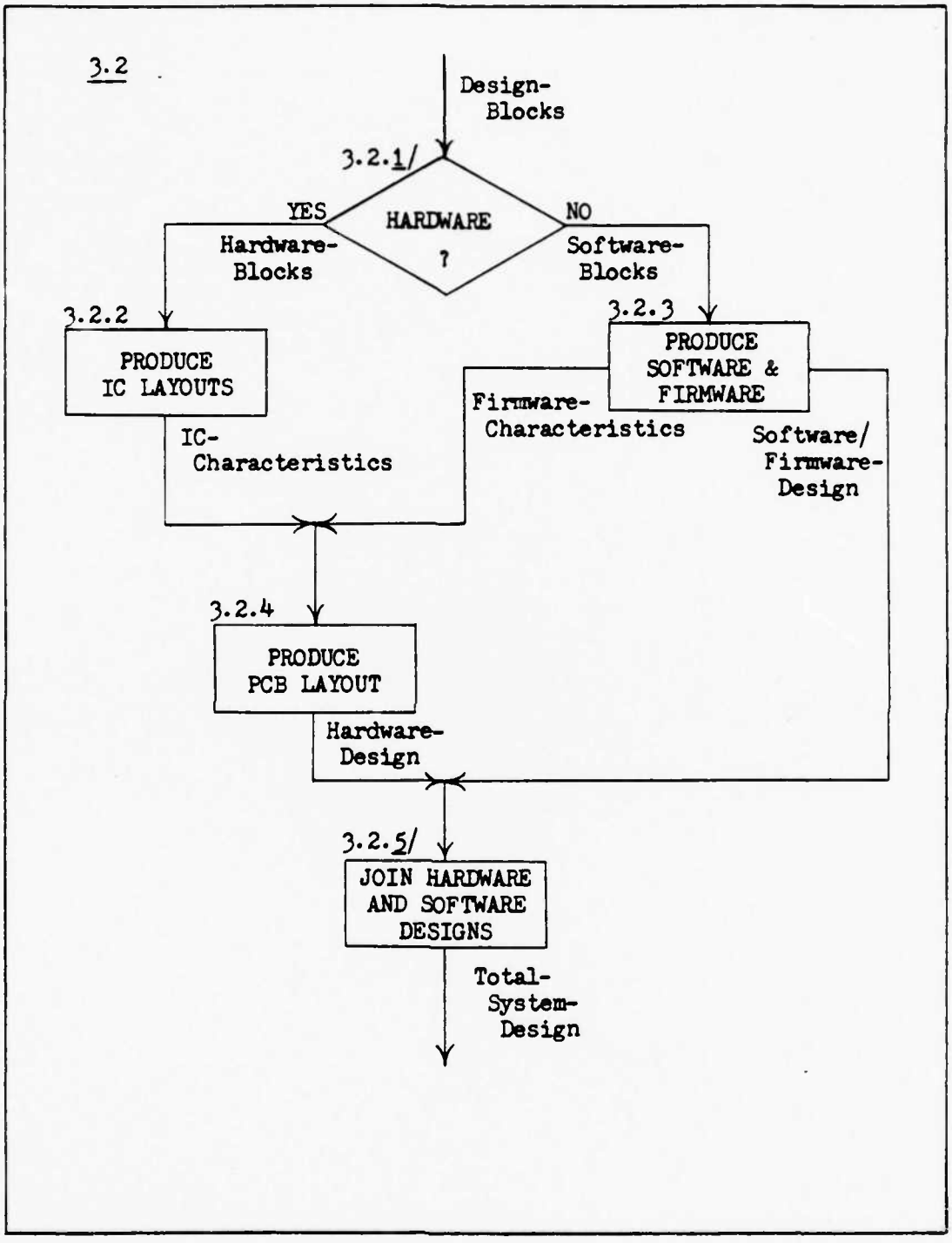



FIG 10. Block 3.2

3.2.1. The first block divides the hardware and software parts of the design, sending them to their respective process blocks. Note that many of the decision blocks in the model function like this one. They may represent iterative processes with several layers and more than one output. As in this case, the flow of data and control may exit the block by two or more paths at the same time. Thus, blocks 3.2.2 and 3.2.3 can operate concurrently.

3.2.2. On the hardware side, the system prepares a layout of any newly designed IC's (Figure 11).

3.2.2.1. First, the blocks which can be realized by existing IC's are separated.

3.2.2.2. The new IC blocks are transformed into IC layouts by the placement-and-routing programs. MP2D (Ref 28) is a prime example of such a tool. This area has received wide attention in recent years. Reitmeyer describes several systems under development, some of which are extensions to MP2D (Ref 3). Tobias discusses several layout methods and highlights work being done at Caltech on "Bristle Blocks, a silicon compiler" (Ref 72:98-99). (See references 73, 74, 69, and 12 for details of the system.) Many other layout systems are also being developed (see Refs 57, 75-79, and 92), and a project is under way at AFIT to apply artificial intelligence principles to the layout problem (Ref 86). The MADAM project used MP2D to run a sample layout for its hardware stack design.

3.2.2.3. After placement-and-routing, the layouts for the new IC's can be examined by the user at a graphics terminal. At this point, the user may make changes to the layout through the interactive graphics system. Notable examples of such tools are "systems offered by Calma,

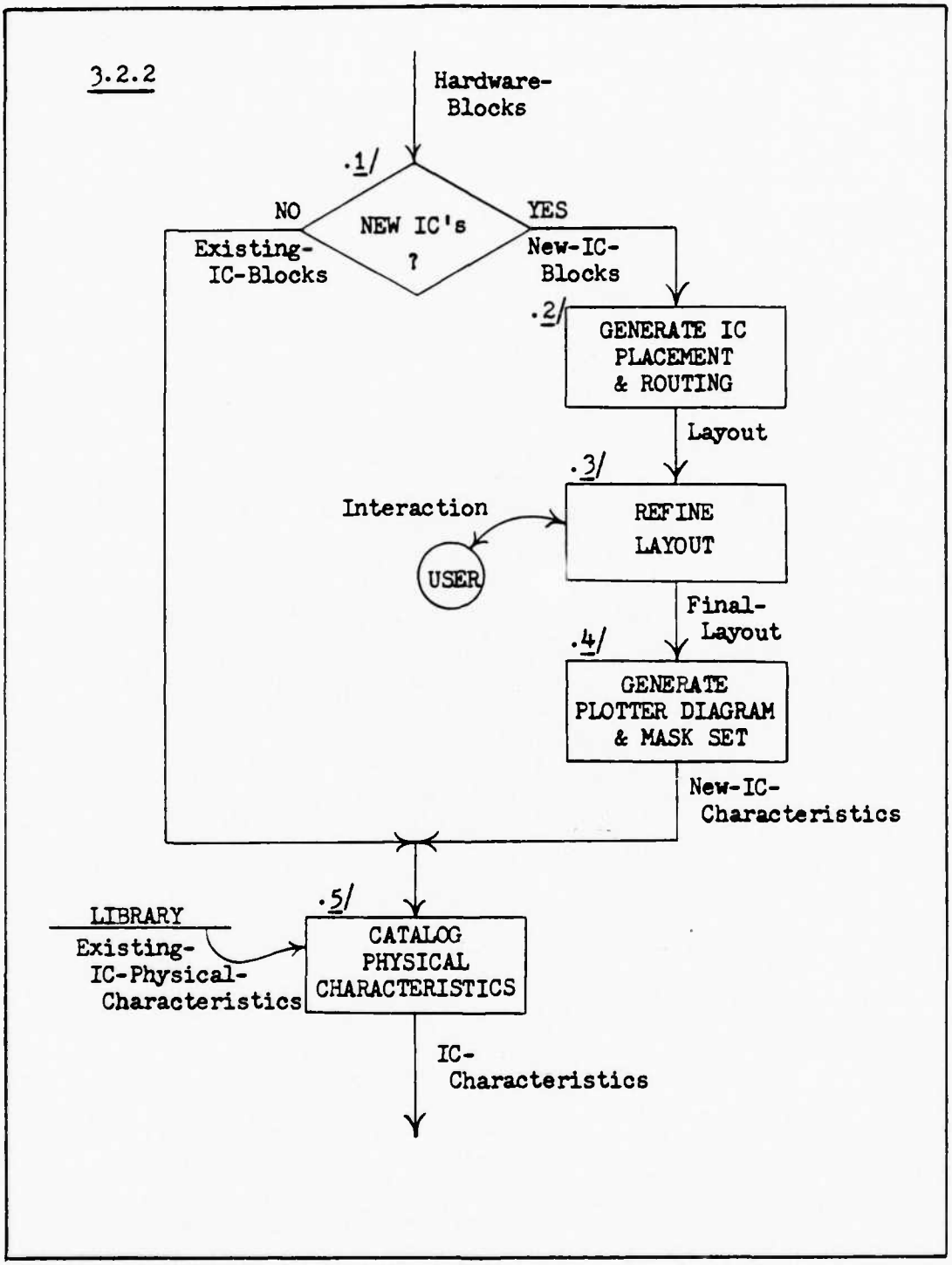



FIG 11. Block 3.2.2

Applicon, and Computer Vision" (Ref 72:98). In addition to being able to analyze and modify the layouts generated by Super-CAD, designers may also "directly draw . . . layout patterns on [the] CRT display" (Ref 72:98) if they so desire. This permits simpler designs to be input to Super-CAD to, for example, develop a mask set or plotter diagram. An additional tool here is the Graphics Work Station under development at AFIT (Ref 58). An Applicon was available for the MADAM project and could have been used to examine and modify the stack design, if necessary.

3.2.2.4. After the layout is finalized, the system will generate plotter diagrams and a set of masks for the new IC's. The Zynetics Flatbed Plotter is an example tool for drawing the diagrams (Ref 25). Also, several projects are underway to enhance layout drafting for mask set production; for example, LTX (Ref 80). Actual production at the present time is largely a manual process. Work is needed in this area to draw it completely into the Super-CAD system. At this point in the MADAM example, a layout diagram of the hardware stack was made on the Zynetics plotter.

3.2.2.5. The final block of Figure 11 catalogs the physical characteristics (pin-outs, dimensions, etc.) of both new and existing IC's (right and left paths from 3.2.2.1) for use in later PCB design. This information comes from the database libraries for the existing chips and from the final layout design for the new chips.

Of course, the user may not wish to go on to circuit board design. With Super-CAD he may chose to terminate the design process here at block 3.2.2 and receive his IC design and a mask set as output. Alternately, a designer could chose to start using the system at this

location. Working through the Executive, he could enter the characteristics of a set of chips at block 3.2.2.5 for designing a complete circuit board.

The experiences in the MADAM project point out how important this area of IC layout will be in Super-CAD. Only a sample layout for the hardware stack was run because the CAD system could not support a design of MADAM's size. Logic 4 simulations could not be performed with more than a couple of modules at a time (see the Implementation section below), and MP2D was similarly limited (Ref 25). Super-CAD will overcome these deficiencies and be able to handle complete VLSI designs.

3.2.3. Returning to Figure 10, on the software side the system proceeds to produce the software portion of the design. As Super-CAD is currently planned, software design is essentially conducted in conjunction with the hardware design during the entire Implementation stage. This is facilitated by the use of existing families of IC's to fill the requirements of the problem. At the same time, existing software modules to operate those IC's will be available in the database. As families of IC's are selected, software modules will also be selected and modified as necessary to interface with the whole system and satisfy specification requirements. Also, it is possible to assign different functions of the problem specification to either hardware, software, or firmware, and the decision can often be delayed (Ref 54:332) down to the logic level of abstraction. When suitable software modules are not available, Super-CAD will design them (or assist in their design) during the logic design portion of Implementation.

Once all modules are complete, and the hardware and software designs have been split up, block 3.2.3 finalizes the software (Figure 12).

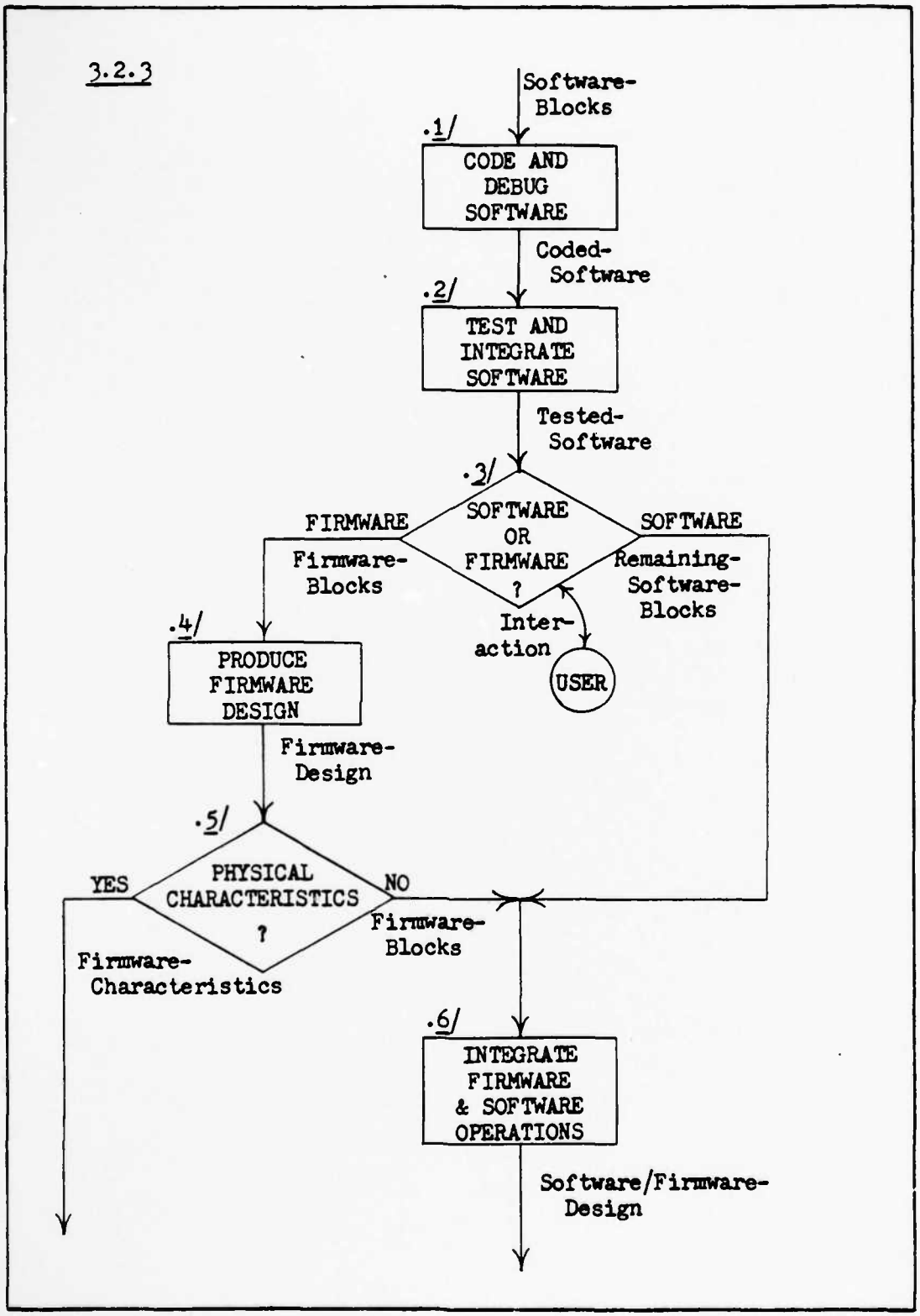



FIG 12. Block 3.2.3

3.2.3.1. Under current Software Engineering techniques, system software follows a software development cycle (Refs 81-84 and 127). Super-CAD will handle the requirements and design portions of the cycle in the Specification and Implementation stages. Existing software modules selected during Implementation will already have been coded and tested. Any new or modified modules, plus the portions of the software design that put it all together to meet the specification, will be handled by this block and the one following. Coding and debugging is still a predominantly manual process, but Irvine and Brackett describe a Software Engineering Facility (SEF) which could lead to an effective Super-CAD tool for automating it (Ref 85).

3.2.3.2. Test and integration of the new code is conducted next, and the SEF would be of use here, also. For Super-CAD to be effective, other automation tools will have to be developed in this area. A current AFIT project will create an automated software development environment, integrating available software engineering tools and techniques and developing new ones (Ref 121). When complete, it will be an important part of Super-CAD. In addition, Super-CAD should ultimately provide an interactive facility to aid designers who want to create their own software.

The MADAM project did not originally include software. If future projects were to complete the MADAM design and approach the software aspects, these last two blocks of Super-CAD could be extremely helpful in producing the software to run the machine.

3.2.3.3. An important decision is whether to implement software in firmware or not. Firmware is computer "circuitry which performs the functions of program instructions" (Ref 131:206). It can be

thought of as "part hardware and part software": "microprograms" that are stored permanently in ROM (read-only memory) IC's (Ref 132:2-33). Thus, firmware "generally refers to software that has been made operationally permanent by storing it in a type of hardware". It allows "many repetitive tasks" to be "hard wired" into the computer for greater efficiency. (Ref 131:207) This block of the model will, with the interactive help of the user, separate the portions of software to be located in firmware (e.g., programs for embedded computer systems in aircraft).

3.2.3.4. The blocks that are to become firmware are then formally organized into the firmware design. Special tools must be created to accomplish this task.

3.2.3.5. Next, the physical characteristics of the firmware IC's will be cataloged.

3.2.3.6. The firmware blocks will rejoin the remaining software blocks for the full software/firmware design.

3.2.4. From Figure 10, the circuit board layout block is expanded in Figure 13.

3.2.4.1. The first step structures the data previously cataloged on the characteristics of the hardware and firmware IC's, for use by the layout program.

3.2.4.2. Then the PCB layout is generated. Soukup likens this problem to that of an IC layout (Ref 57:1281). This is another area where many current CAD tools may be brought into Super-CAD. Some examples are NOMAD (Refs 87 and 88), DASLL (Ref 89), and BRAIN (Ref 90). Also, a research project on PCB routing is in progress at AFIT (Ref 91).

3.2.4.3. The next block is similar to 3.2.2.4 for IC layouts,

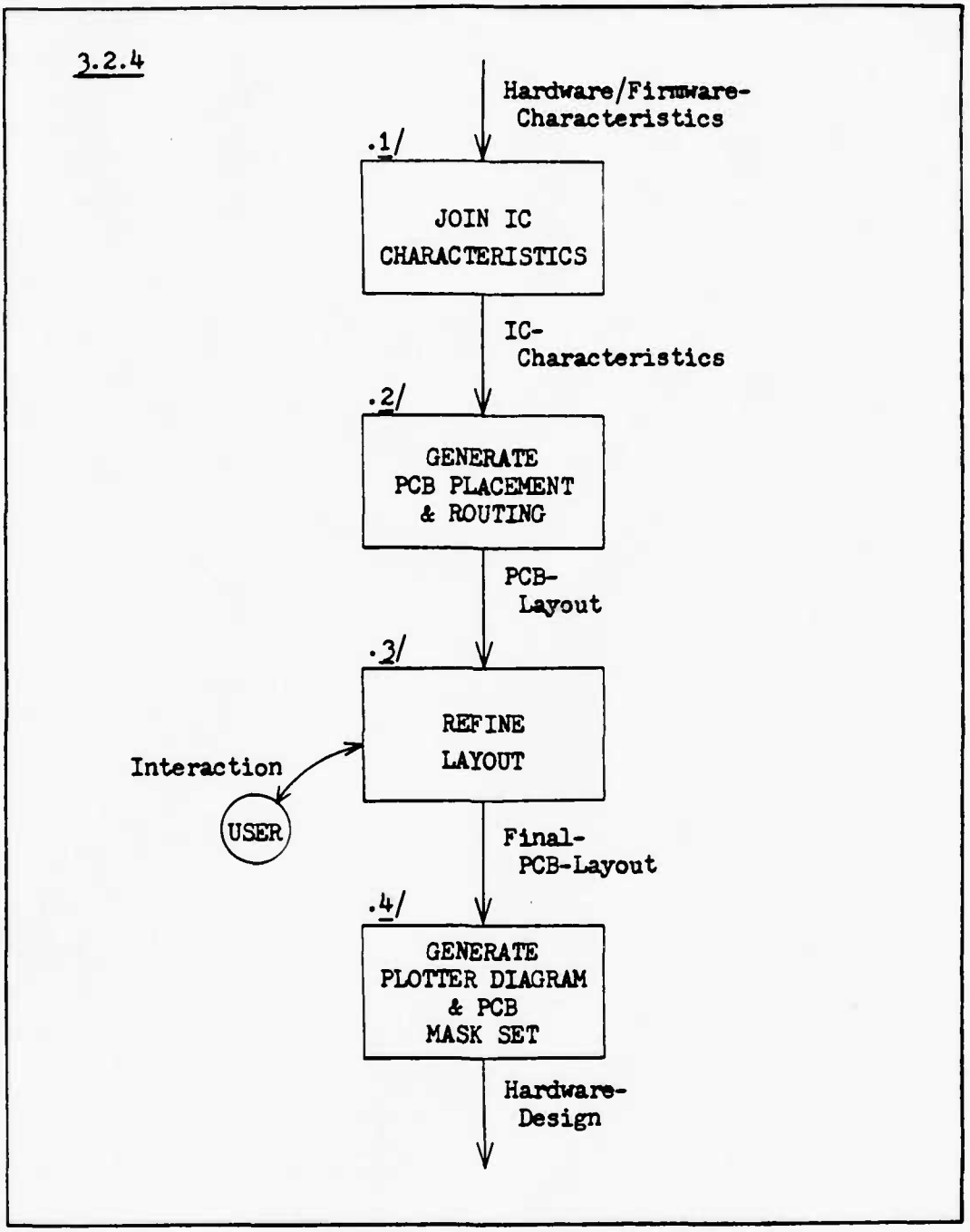



FIG 13. Block 3.2.4

where the user can refine or modify the final layout. The same tools apply.

3.2.4.4. This step is also similar to an earlier one for IC's, 3.2.2.7. The same plotting tools are used, and--as before--more work is needed to automate mask set production.

3.2.5. The final block in Figure 10 is where the complete hardware and software designs are joined together to provide the total system package for the user. The fact that this block is not decomposed beyond Figure 10 may be misleading. It will incorporate many end-of-design operations, such as formatting the hardware and software portions for output, and requires future work to be defined fully.

This completes the discussion of block 3, the Realization stage. We have seen how the development of MADAM paralleled many of the steps given in this stage. That project could have benefitted significantly from a system like Super-CAD. The MADAM example is used with greater detail in the next section, which examines the Implementation stage.

Implementation. (Block 2, Figure 7) The major emphasis of this project is on Implementation. Thus, this part of the Super-CAD model is the most developed. Figure 14 shows Implementation broken up into two main blocks. It is assumed that the Specification stage (block 1, to be described later) has produced a behavioral specification as input to this stage. Partitioning of the problem into a suitable specification will be discussed in the Specification section below. The important thing is that the specification must be structured enough to allow sub-set partitioning for use in the Implementation stage.

2.1. Super-CAD attempts to complete a design at as high a level of abstraction as possible. Chapter I showed that this can simplify the

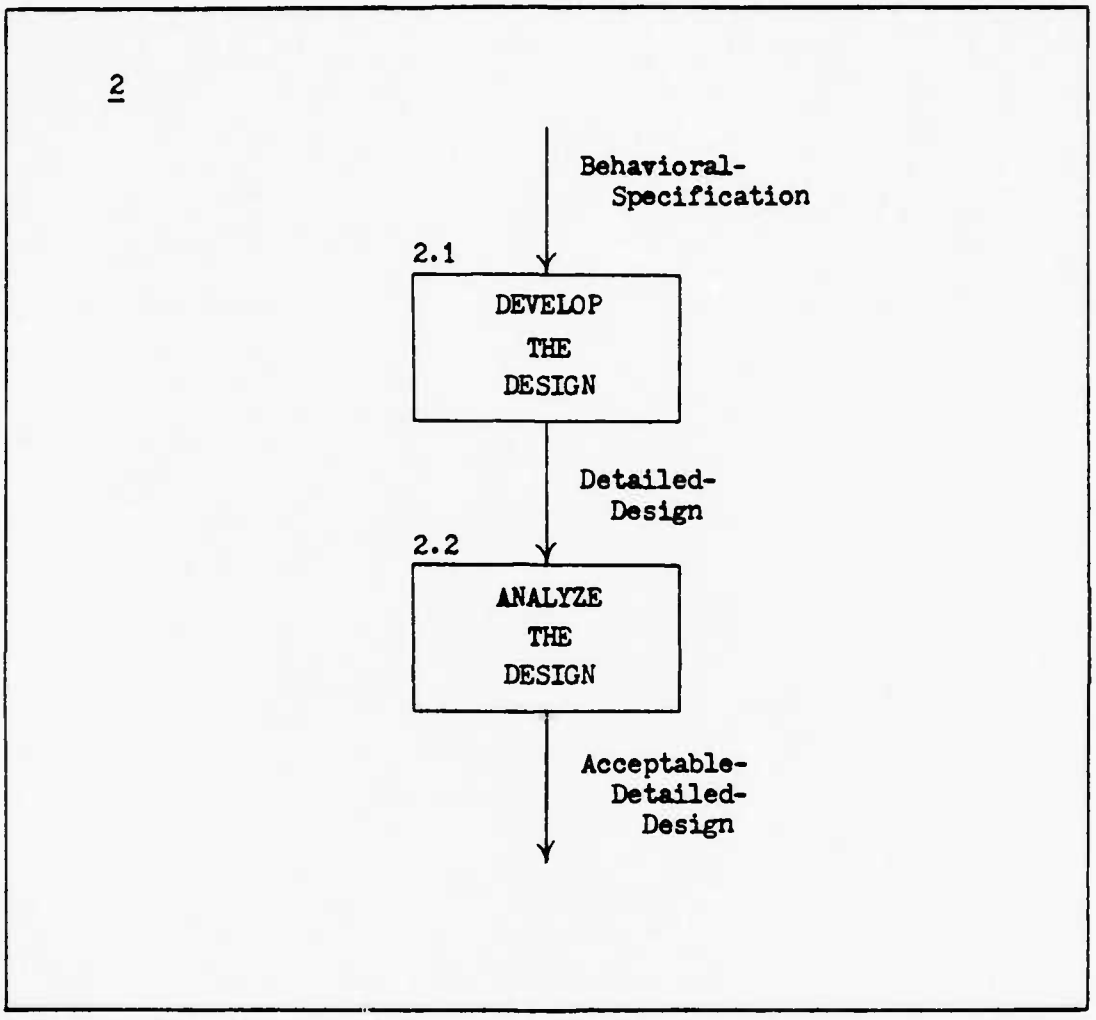



FIG 14. Block 2 - Implementation

design process--by masking many lower level details--and produce improved designs more efficiently. Referring to Figure 15, the problem is attacked first at the Behavioral level. If all or part of it cannot be solved with existing implementations at that level, Super-CAD takes what is left to the Functional level. If any part of the problem is still not implemented, it goes to the Logical level for final implementation. At any step, whenever all subsets of the specification have been satisfied, the design is finisned. Then all that remains is to interface the constituent implementations, including any bits-and-pieces that must be added, to complete the design.

The result of this approach is a top-down design, with implementations at the highest possible level. As discussed earlier, this is essential if VLSI/VHSIC design is to be effective (Ref 57:1302). The strength of this system is that it satisfies as many specification subsets as possible by mapping them to appropriate existing implementations in the database libraries. It filters down to the level where enough details are present to complete the design, including, in some cases, synthesis of new circuits at the logic level.

The process is supported by libraries of components at all levels (Ref 52:92), with new families being added as they become available. The flexibility of the system is further enhanced by allowing the user to choose the mode of design. As we shall soon see, the normal mode is highly interactive. Working through the Executive program (Chapter II) the designer can help the computer to solve difficult parts of the design; for example, where existing implementations are only "close" to the specification requirements. On the other hand, it will be possible for him to prohibit any close solutions and force the design all the way

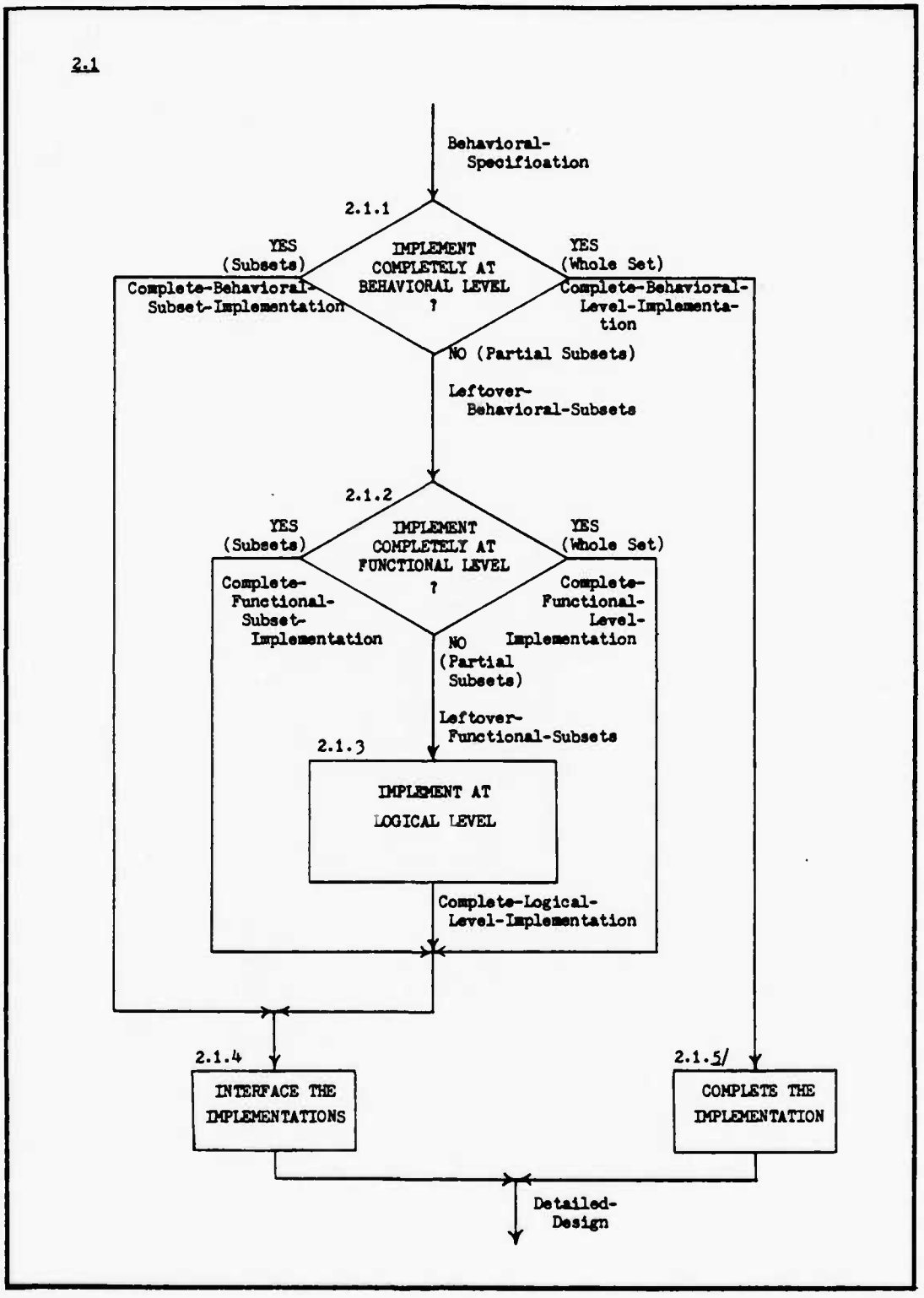



FIG 15. Block 2.1

to the Logical level. This might also be accomplished by allowing him to select only the logic library so that the specification automatically falls through to total synthesis at the logic design level.

Another choice that would deny any close implementations would be the automatic mode. In this case, the designer could input his requirements and wait to see if Super-CAD can complete a design without his help. Then if it failed, he could try again interactively.

A number of tools are under development that can be applied to digital design at the level of block 2.1. While most are not compatible with Super-CAD, some of their techniques are important and need to be considered as Super-CAD is further developed. (See Refs 93-100, 116, and 35.) Some of these tools are Hardware Description or Register-Transfer Languages. It should be possible to allow a user to bypass Super-CAD's Specification stage and enter his requirements in any one of these languages. The system would need suitable translation programs to transform the input for use by the blocks within 2.1.

2.1.1. The Behavioral level is divided into two portions (Figure 16). The first attempts to implement the complete behavioral specification at one time, by mapping the whole specification to an available implementation or set of implementations.

2.1.1.1. Figure 17 shows this block decomposed to its lowest level for this report. Notice that, because of the length of the numbers at this level, only the final digit is shown on each individual block. Also, since this is the final level of division--with many more blocks in later figures--the "_/" convention is no longer followed. Whenever the abbreviated numbers are used, no further decompositions are present. The abbreviated numbers will also be used for the paragraph

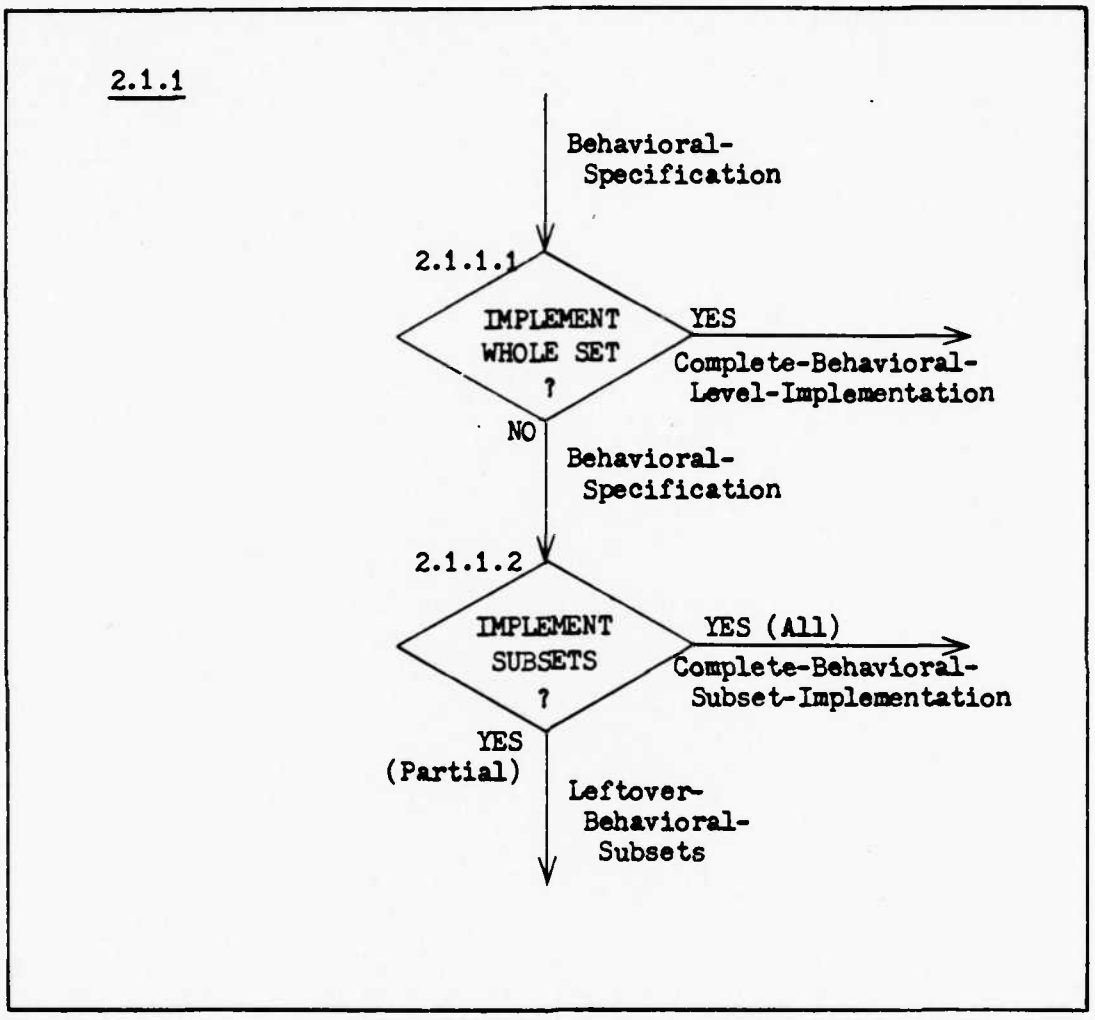



FIG 16. Block 2.1.1

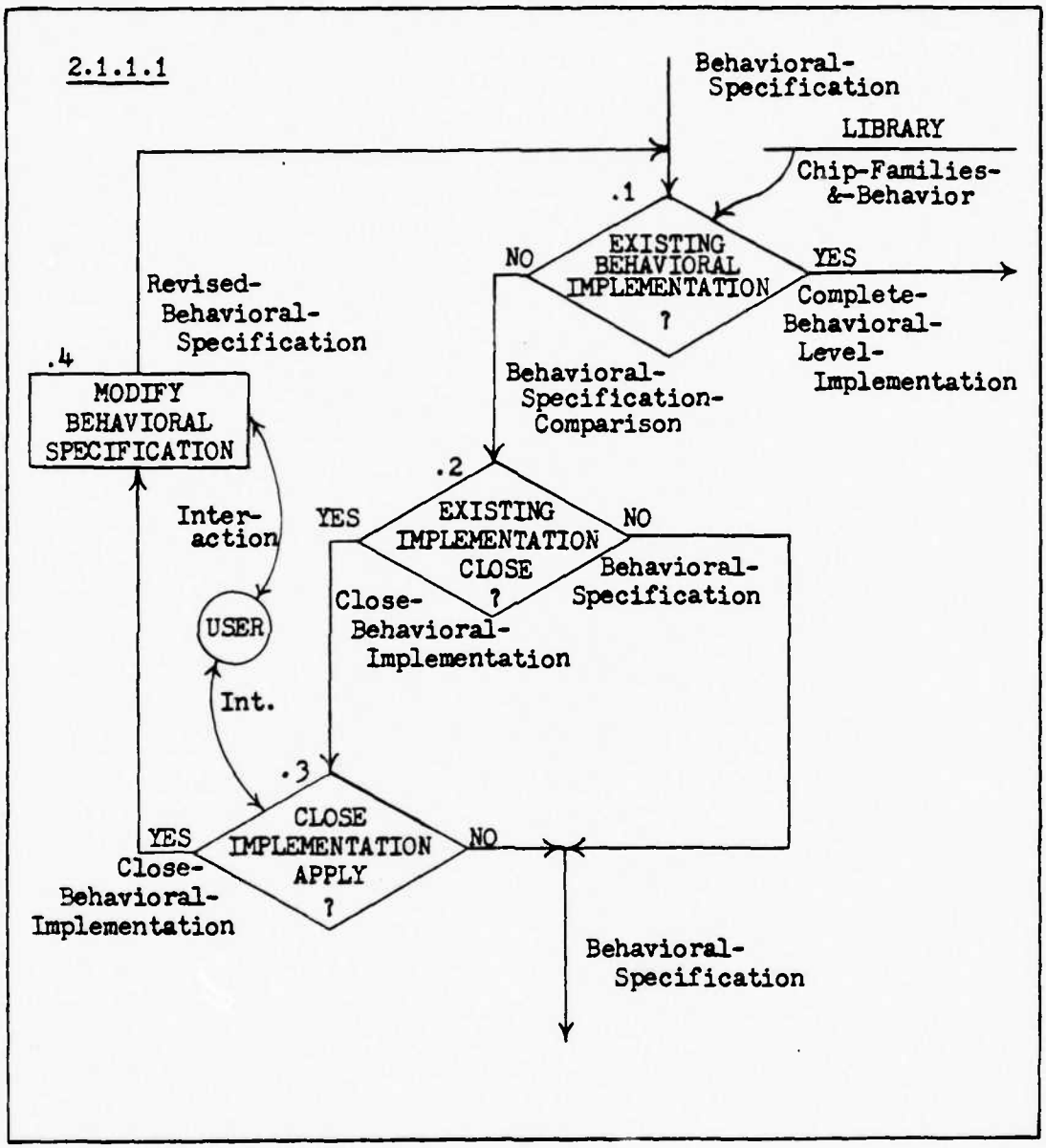



FIG 17. Block 2.1.1.1

headings.

.1. The first block is responsible for comparing the behavioral specification with the behavioral libraries to determine if families of circuits can be found to implement the complete specification. If so, the design is complete and it is sent on to block 2.1.5 (Figure 15). If not, the comparison goes to the next block.

In the MADAM microprocessor example, an existing processor family might be able to satisfy all of the specifications and perform the Ada subset. Of course, the user would still have the option of rejecting the implementation if he was not satisfied with its performance characteristics, possibly in areas he had failed to emphasize during Specification.

.2. In the next block, a routine determines if any of the library solutions are close enough to the specification for possible use, depending on the desires of the user and his ability to modify the specification. If not, the specification is sent out to block 2.1.1.2 (Figure 16).

.3. But if a close solution is available, the block interacts with the user to decide if it applies. (In the "automatic" mode, "no" would be the only option here.) Again, if the answer is "no", the specification is sent to 2.1.1.2.

.4. If the answer is "yes", however, the user can modify the specification interactively, and send it back for another try.

An example of a close implementation for MADAM might be where all but one of the instructions could be performed (e.g., a stack manipulation instruction (Ref 25: Part I, 9)). It would be up to the user to decide if that instruction is really necessary or could be accomplished

a different way.

2.1.1.2. If the system cannot satisfy the whole specification, 2.1.1.2 attempts to implement subsets (Figure 18).

.1. Partitioning of the specification is discussed below under the Specification block, and Super-CAD will perform the partitioning in that stage. (An alternative would be to accomplish it here at the beginning of 2.1.1.2.) In MADAM the partitioning might be strictly between the individual instructions. Block .1 divides the specification according to the partitions and assigns indexes through a simple "FOR" loop.

.2. This block initializes the index to 1.

.3. Next, the currently indexed subset is designated and sent to the following block.

.4. The subset is compared to existing circuit behaviors to find an implementation. In the case of a MADAM behavioral subset made up of one or more individual instructions, an implementation might be available to perform those instructions.

.5. If one is found, it is identified (e.g., by entering an identifier in a table of implementations).

.6. If not, the subset comparison is examined for a solution that is close. As before, a close solution is one that meets most, but not all, of the specifications of the subset. For example, the MADAM instructions for manipulating the stack architecture may not completely match up with any available implementations. But it might be worthwhile to consider a change in the requirements to benefit from an existing behavioral implementation.

.7. If none is found, the subset is identified as not

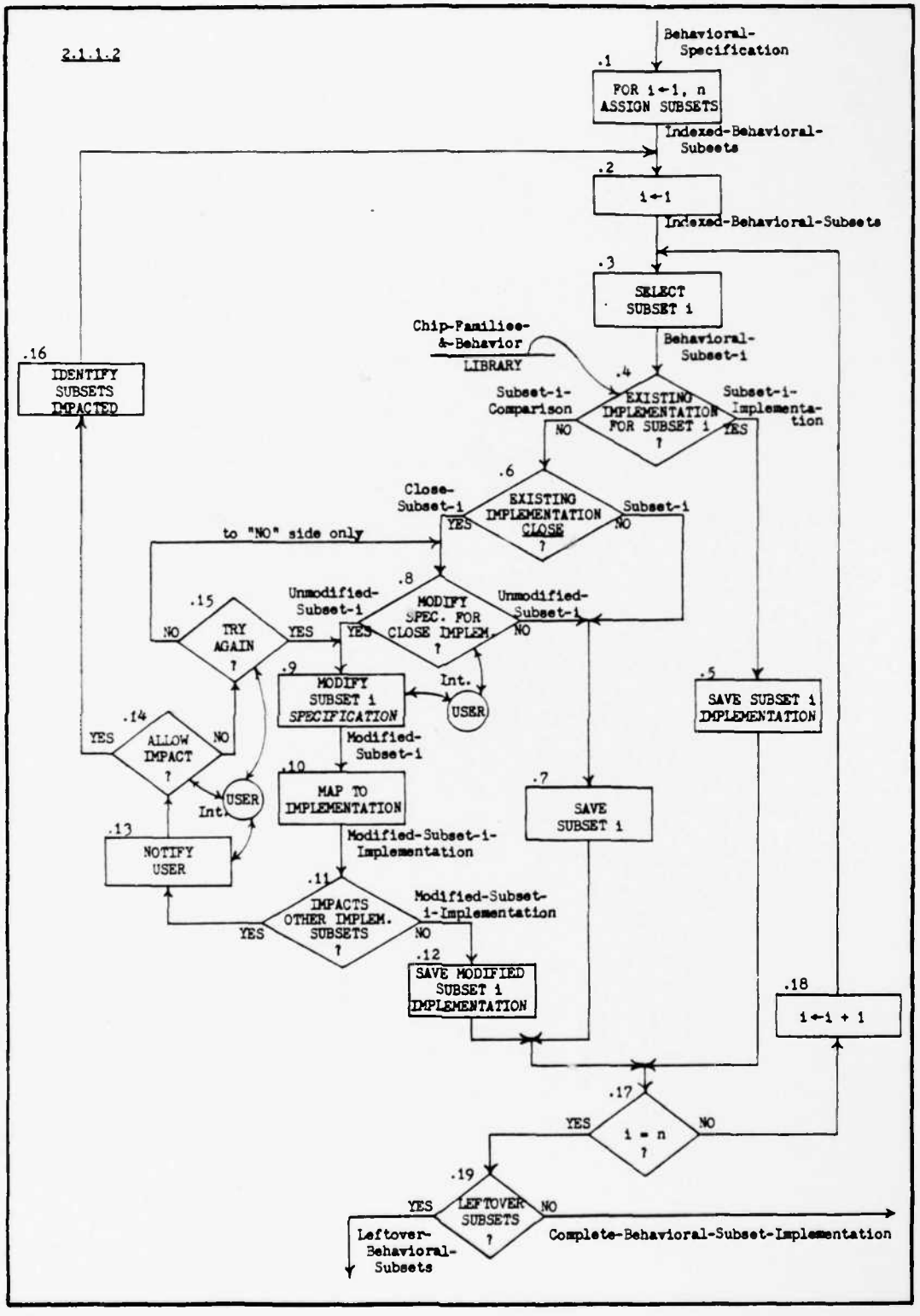



FIG 18. Block 2.1.1.2

implemented. (Again, a special table could be used.)

.8. On the other hand, when a close solution is found, Super-CAD asks the user if he can modify the specification for that subset to be implemented. A "no" answer sends the subset to block .7 to be identified as above.

.9. A "yes" allows the user to modify the specification. With a MADAM stack instruction, he could revise the requirements to fit the existing solution.

.10. Then the system maps the subset to the implementation.

.11. At this point it is important to determine if the modification has affected any previously implemented subsets. For example, modifying one MADAM stack instruction could prevent another one, already implemented, from functioning properly.

.12. If not, the new implementation is saved, as before.

.13. However, if there is an impact, the user is notified.

.14. He can then decide if the impact should be permitted.

.15. If not, Super-CAD gives him the choice of trying again to modify the specification for the subset at block .9. If he declines, control goes to the "no" side of block .8 to keep track of the unimplemented subset at .7.

An interesting enhancement here would be to insert a loop where the user could attempt to help Super-CAD modify the specification partitioning to arrive at different subsets. In the MADAM example, perhaps the three interrelated stack manipulation instructions could all be combined into a single subset. This concept is left for future work.

.16. On the "yes" side of .14, if the user allows other implemented subsets to be affected, they must be identified. A routine

flags those particular subsets. Super-CAD then returns to the beginning of the loop to attempt to reimplement those that have been flagged. The others will fall right back through, since they do not require reworking. Thus, the MADAM stack instruction specifications might each be modified to allow close implementations where they will operate together properly.

This process will be effective precisely because, for any subset affected by the changes to another, the system will immediately redo that implementation. While this may require a number of iterations, each time the system goes through it, more of the specification will be satisfied, with a decreasing trend in the amount of computer work required. It is the user who decides if any changes should be made or impacts on other subsets allowed. As before, the automatic mode will bypass any attempt to rework a close implementation.

At the same time, Super-CAD must anticipate the possibility of "thrashing", where one modification leads to another, which leads to another, etc. It would be up to the user to avoid this and choose to send subsets to the next level for implementation. However, Super-CAD may have to suggest it to him after he has gone through a certain number of iterations without any progress.

.17. After a particular subset has been processed, this block runs a simple test to see if that was the final subset.

.18. If not, the index is incremented and the process continues for the next subset.

.19. When all subsets have been processed, Super-CAD checks to see if any have not been implemented (e.g., by examining the "unimplemented" table). If none remain, the design is complete and it is

passed on to block 2.1.4 (Figure 15). Otherwise, the leftover behavioral subsets (e.g., any MADAM instructions not yet implemented) are sent to the Functional level.

2.1.1 Summary. Returning to block 2.1.1 on Figure 15, several points should be clarified. The outputs of 2.1.1 have been described in the previous paragraphs. The one on the right is the case where the whole set of specifications can be met by one complete behavioral level implementation. On the left is the situation where the problem is solved completely at this level, but by implementing the specification in subsets. The output at the bottom is for the case where a partial implementation of subsets occurs at the Behavioral level. This includes the situation where none of the subsets could be implemented. Thus, the leftover subsets are passed down to the next level, including the situation where they are all leftover.

Since this part of the model is an area that has never been automated in this way before, there are no existing tools to cite as examples. The Carnegie-Mellon work certainly operates at these levels, but the approach is quite different, as we shall see in the Comparison section. Thus, the tools to support the blocks of Figures 17 and 18 will have to be developed specifically for the Super-CAD system.

2.1.2. The Functional level performs in exactly the same way as the Behavioral level, with one important addition. (See Figure 19.)

2.1.2.1. The addition is this block, which translates the remaining behavioral subsets into a Functional level specification. Although it is not decomposed further here, the block will be made up of several programs necessary to accomplish the translation. Not the least of these would be one to duplicate the partitioning of the original

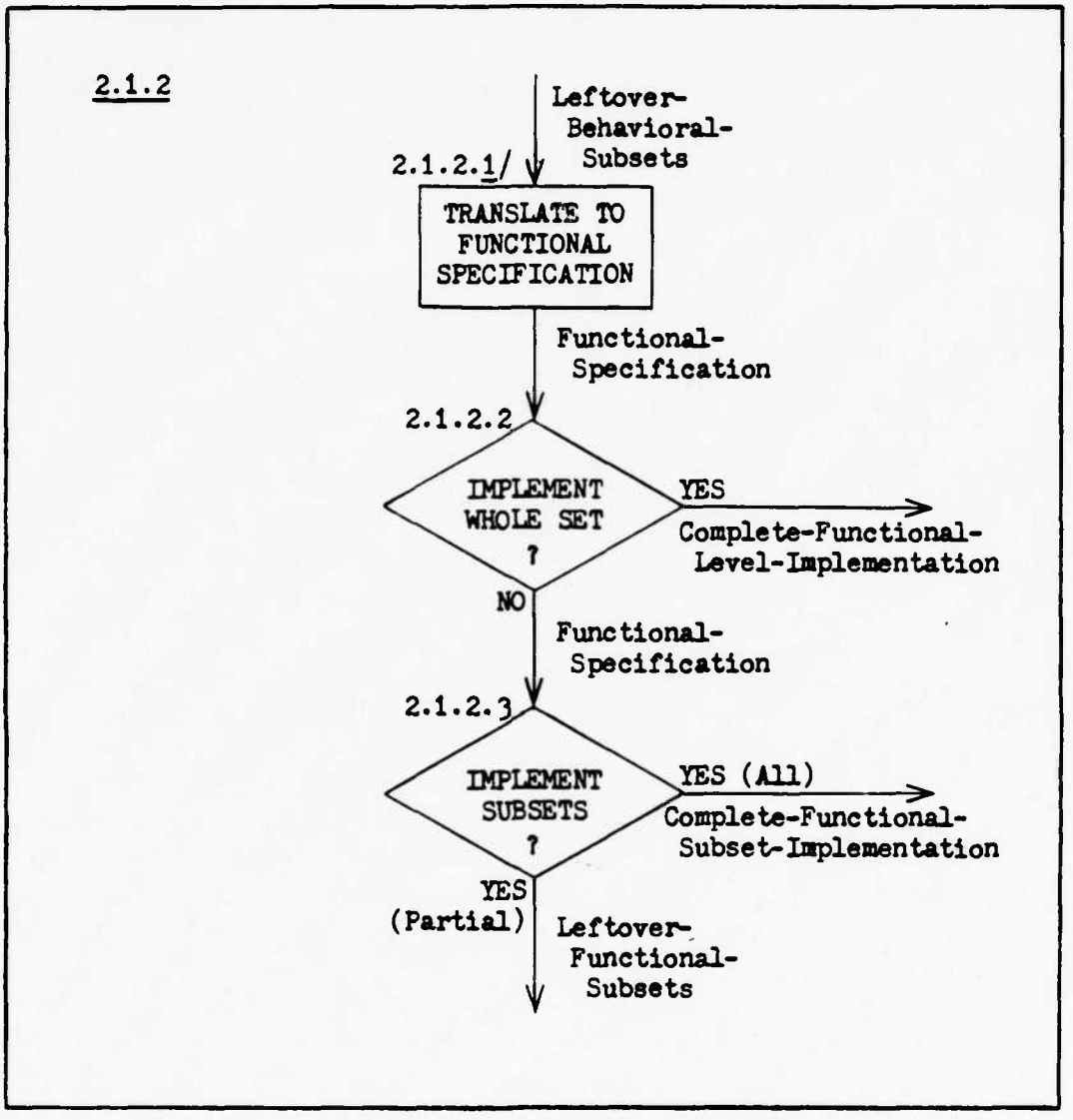



FIG 19. Block 2.1.2

behavioral specification in the Specification stage. Partitioning here could be along the lines of the behavioral subsets, or along other lines to facilitate implementation at this level.

Continuing with the MADAM project example, the Functional level specification might retain the same instruction set partitioning. However, more detail is required, such as a decription of the register-transfers and functional operations necessary to perform the instructions.

2.1.2.2. The next block attempts to implement the whole functional specification, performing operations parallel to block 2.1.1.1. Figure 20 shows these operations. Since this block of the model duplicates Figure 17, the constituent blocks will not be described separately. Refer to the paragraphs under 2.1.1.1 for an explanation of the individual operations.

2.1.2.3. (Figure 19) When the whole functional specification cannot be satisfied with a single implementation, Super-CAD again looks at subsets. As in the previous paragraph, Figure 21 duplicates the operations of 2.1.1.2 in Figure 18. The paragraphs under 2.1.1.2 explain the equivalent operations.

2.1.2 Summary. Back at Figure 15, the block 2.1.2 outputs are similar to those of 2.1.1. The only difference is the one on the right which might have to be interfaced with behavioral subset implementations, and thus goes to 2.1.4 instead of 2.1.5. For instance, some of MADAM's instructions may have been implemented as behavioral subsets and others as functional subsets.

Again at this level most of the tools to support the operations will have to be created. Thomas describes how the Caltech Silicon Compiler

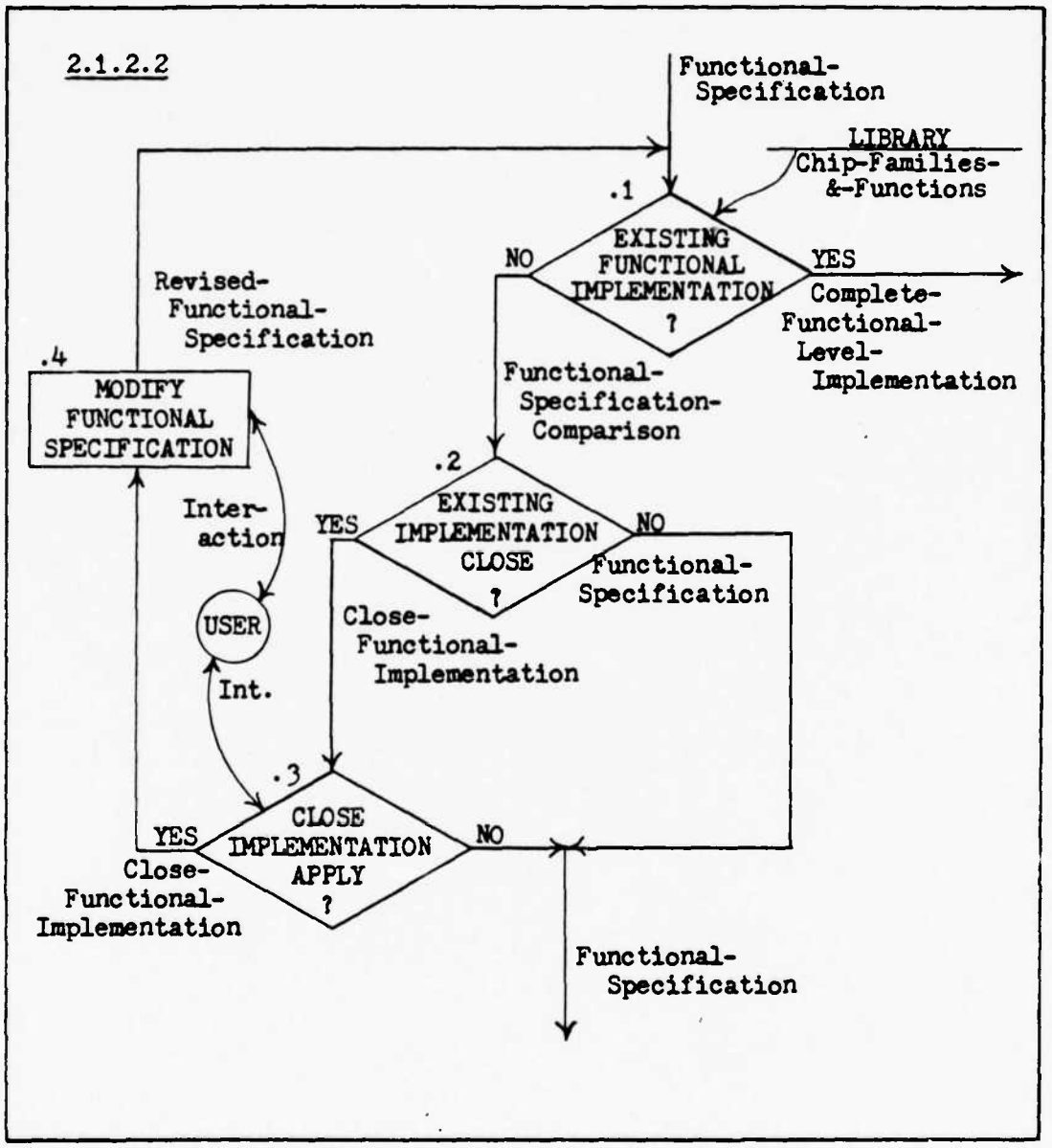



FIG 20. Block 2.1.2.2

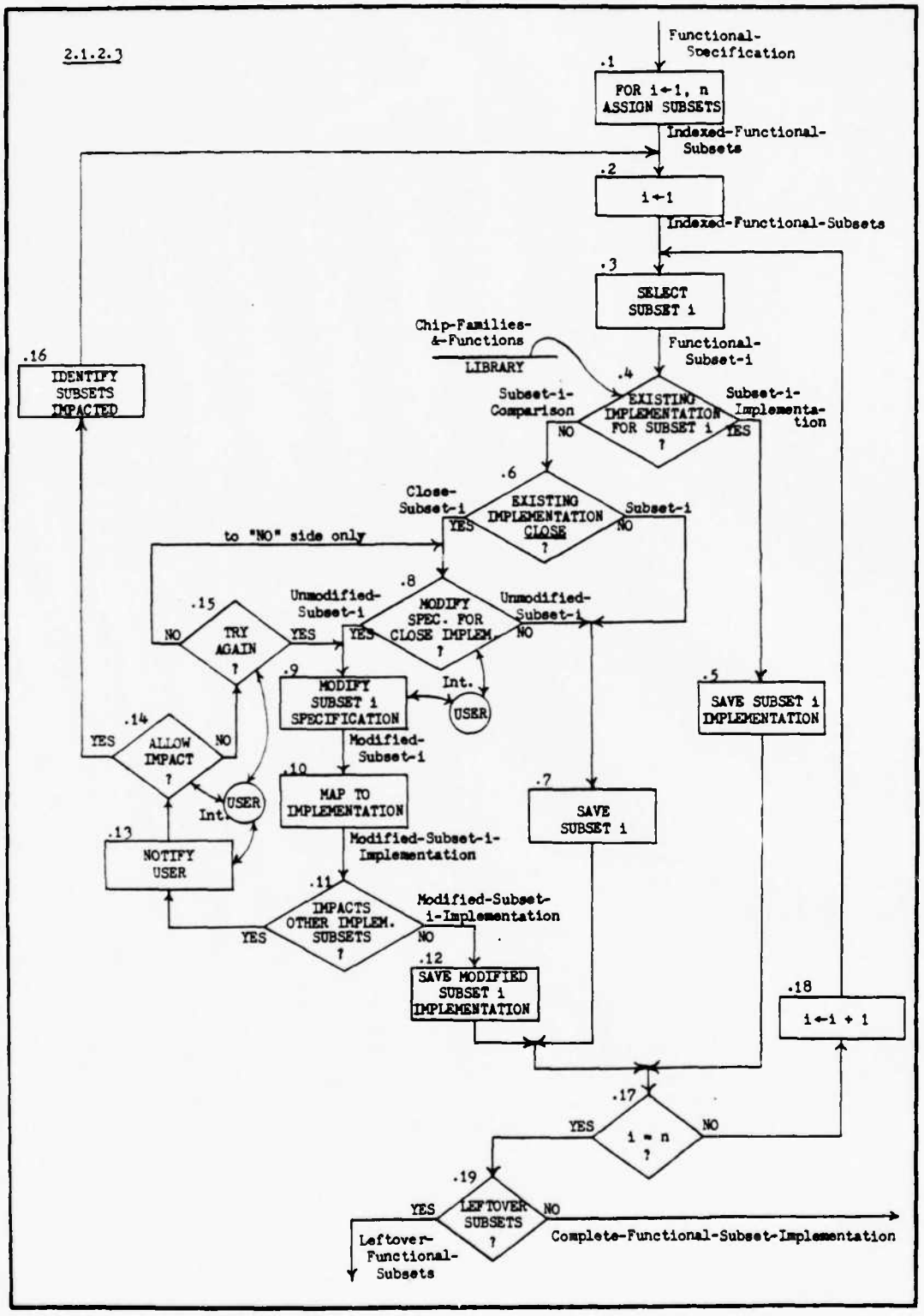



FIG 21. Block 2.1.2.3

operates at the functional level (Ref 12:1207), and some of the techniques might be useable by Super-CAD.

2.1.3. Any remaining subsets at the Functional level go down to the Logical level. Once again the procedures are the same, except that this level has only one output. This is the lowest level, and if a design cannot be completed with existing IC's, an additional block will synthesize new ones (Figure 22).

2.1.3.1. First the leftover functional subsets are translated into a Logical level specification. In the case of the MADAM instructions, specific combinations of registers and designated modules (adder, multiplier, stack controller, etc.) would be described. As before, Super-CAD tries first to find an implementation for the whole specification, and if that fails, seeks subset implementations.

2.1.3.2. See Figure 23 and the explanation for 2.1.1.1.

2.1.3.3. See Figure 24 and the explanation for 2.1.1.2.

2.1.3.4. If any subsets still remain, the logic synthesis block is activated (Figure 25).

.1. First a set of tools partitions the subsets to prepare them for logic design. With the MADAM design, the logical modules would be broken up further into basic building blocks (gates, flip-flops, etc.).

.2. Then logic synthesis programs take the subsets and apply elements from an appropriate cell library in the database to complete the design. For example, the MADAM project, through the Logic 4/MP2D programs, used a CMOS/SOS cell library composed of 44 different gate-level components (Ref 25: Part I, 3-4). This step and the previous one replace the manual logic synthesis actually used in the MADAM

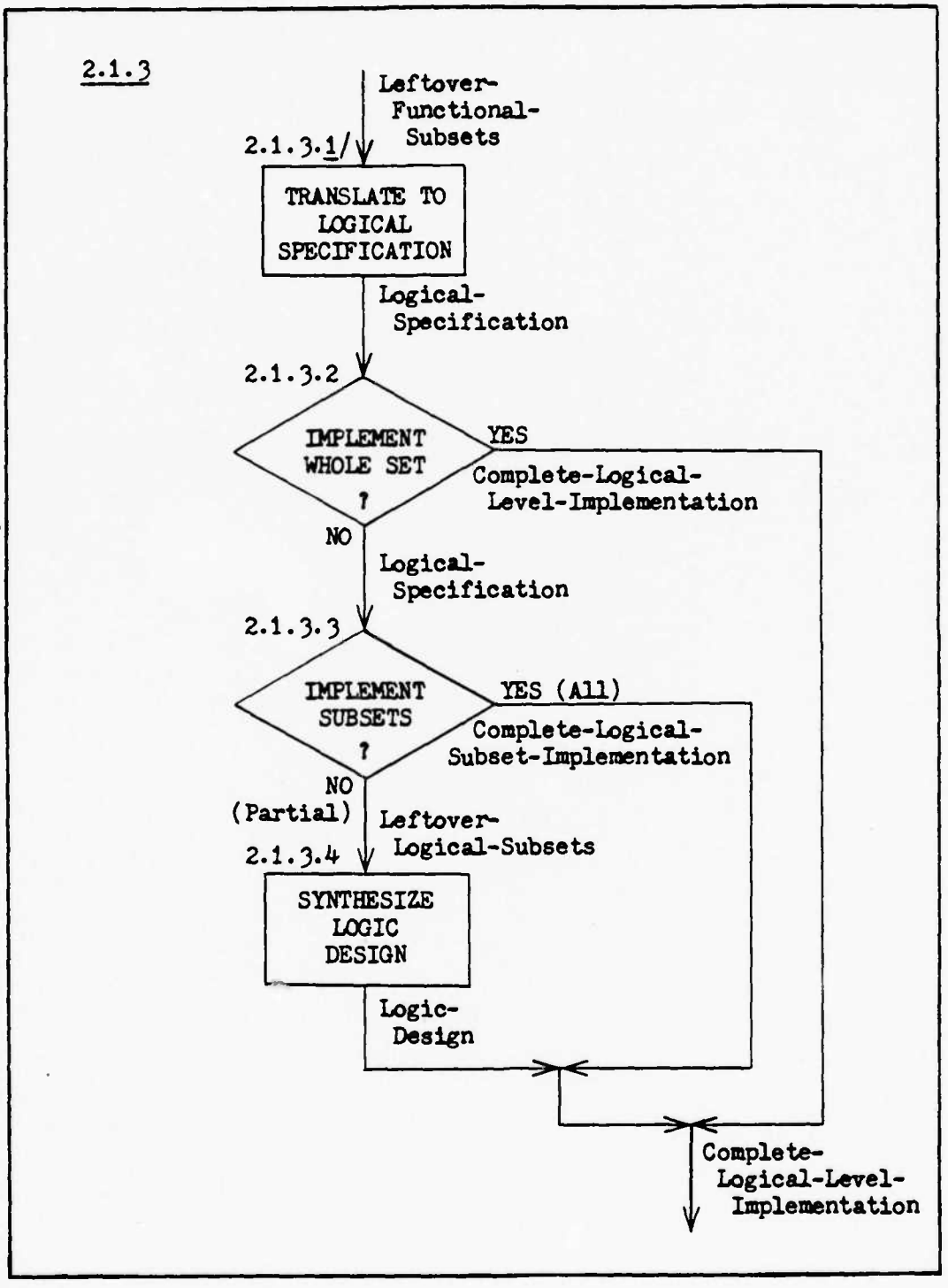



FIG 22. Block 2.1.3

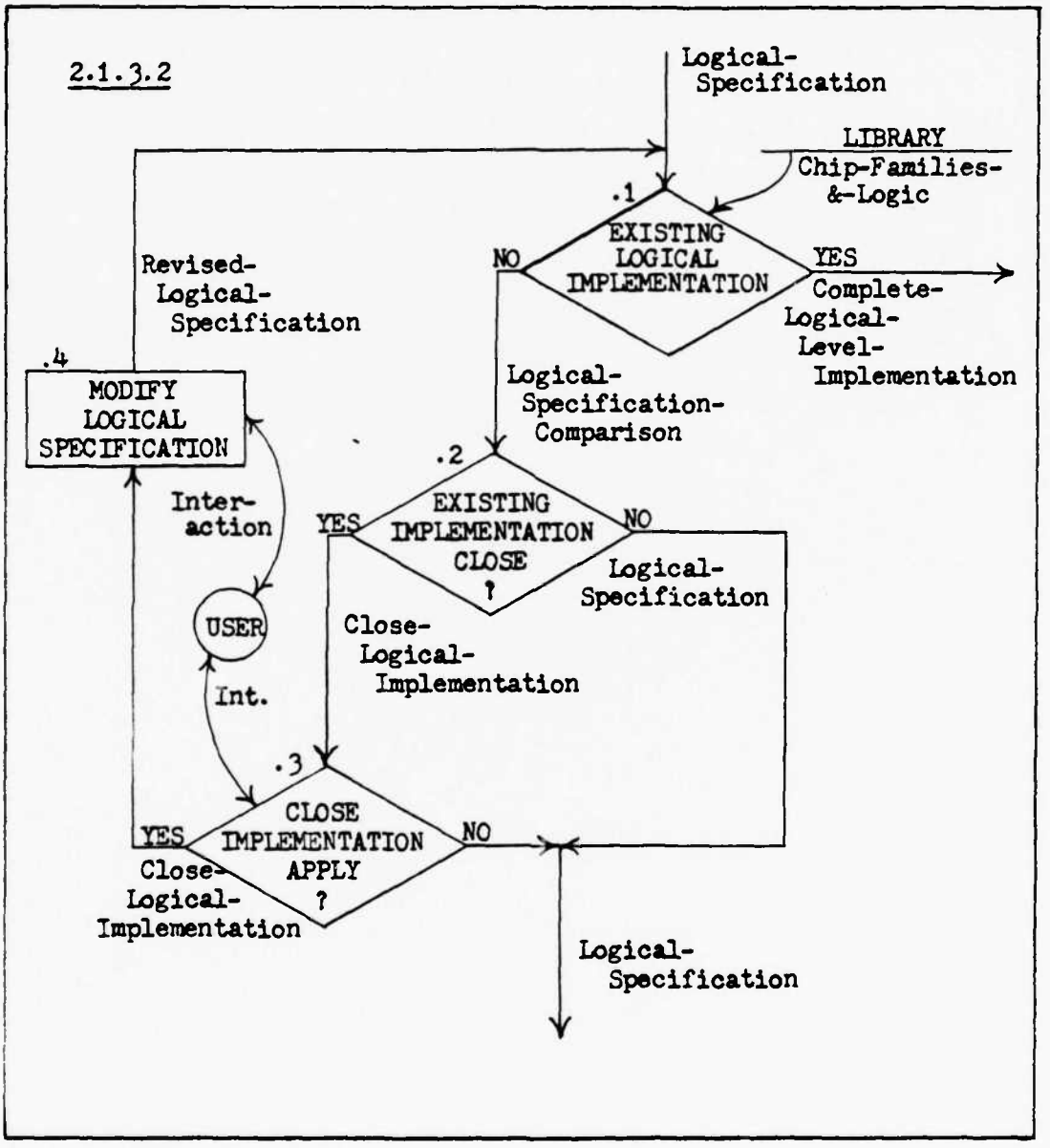



FIG 23. Block 2.1.3.2

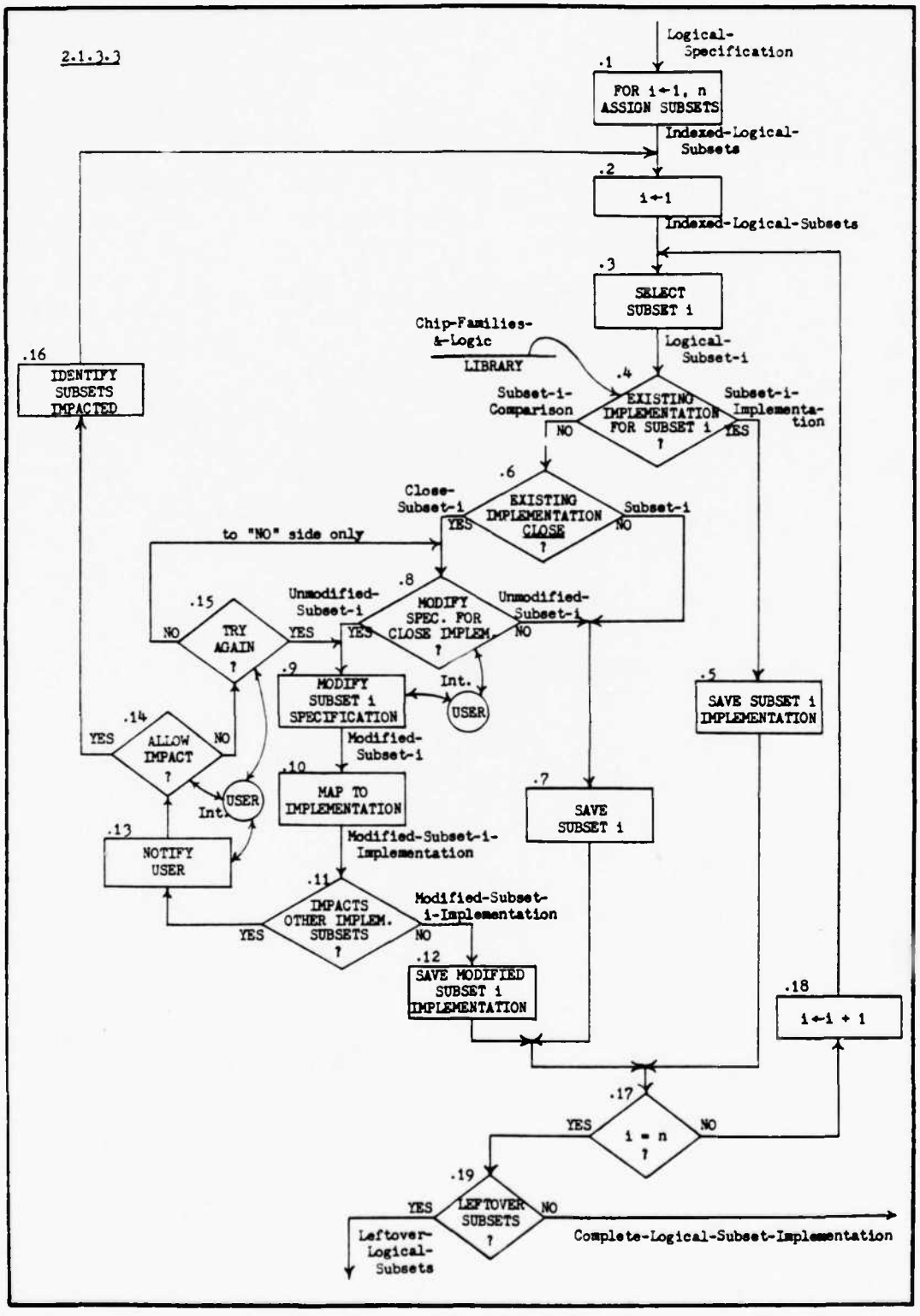



FIG 24. Block 2.1.3.3

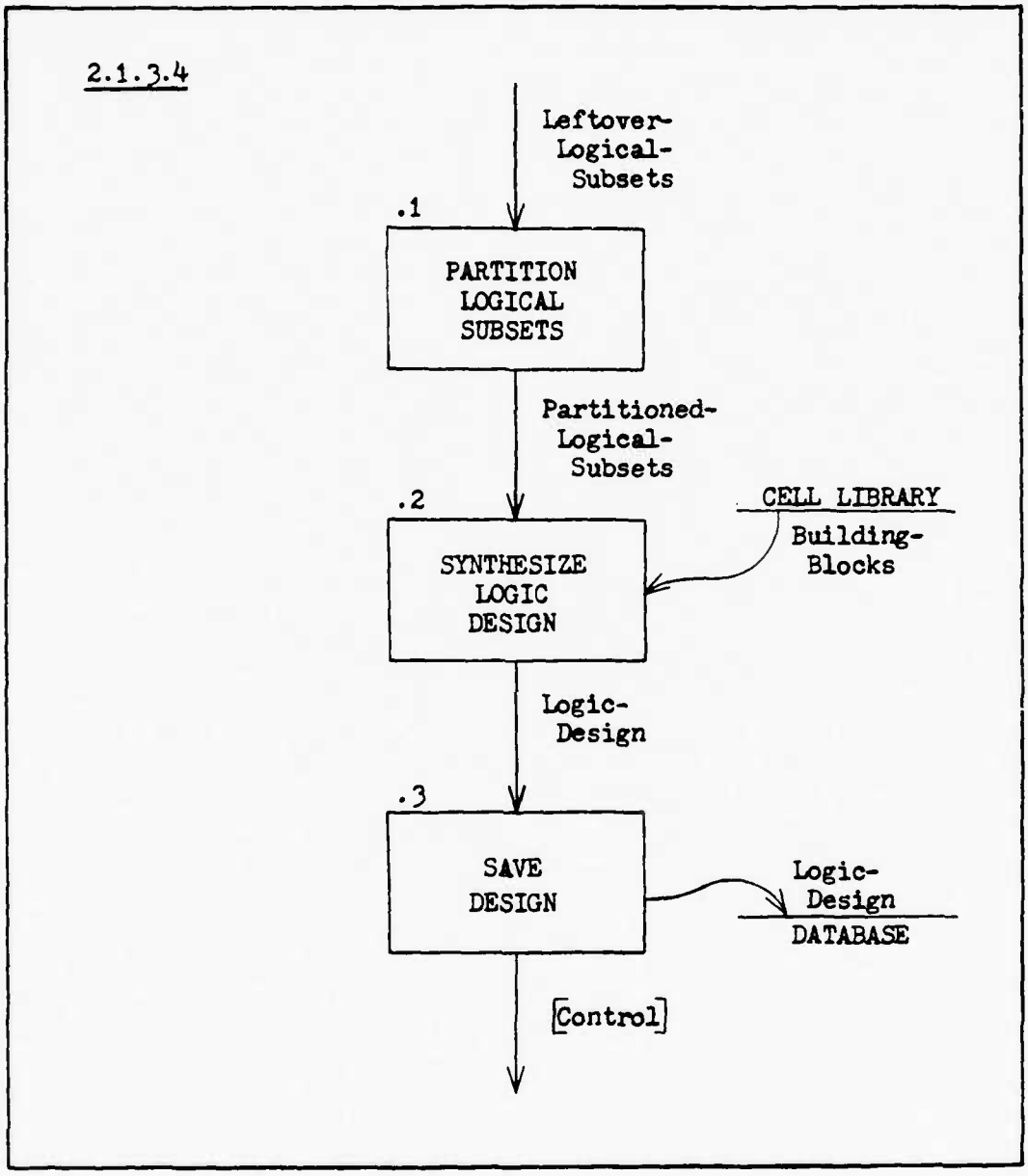



FIG 25. Block 2.1.3.4

design.

.3. Finally, a routine stores the logic designs in the database.

2.1.3 Summary. The last three blocks actually represent a very complex portion of the system. A lot of future work will be required to fill them in. However, tools are available to provide some guidance and support. For example, PLA (Programmable Logic Array) chips and a variation, PAL (Programmable Array Logic), provide a certain amount of flexibility for VLSI by permitting custom programming (Refs 72:92; 7:1194; and 76:658). Thomas describes several logic synthesis aids (Ref 12:1205-1209). For example, ALERT was an early system, and it is further explained by Friedman and Yang (Ref 101). Another was MIMOLA for designing digital processors. (See also Refs 102 and 103.) A highly interactive approach is being pursued at IBM for automatically generating a "detailed, technology-specific implementation" (Refs 104:234 and 105:543). In addition, CARS proposes a "top-down structured approach [for] digital synthesis" (Ref 106:529).

Currently Super-CAD proposes to create new IC designs only at the Logical level. Future work can expand it to encompass the Functional and Behavioral levels as well, especially as new and more sophisticated building blocks are added to the system. Thus, complex modules defined at the higher levels could be part of cell libraries at those levels. New IC designs could be created from such modules without requiring analysis at the logic level. (The logic would be implicit in the modules.)

2.1.4. The next step (Figure 15), after the design has gone through all three levels of abstraction, is to gather together the

implementations from the various levels. It might appear from Figure 15 that this "interface" block is meant to combine all of the arrows that lead to it. However, any one design problem follows only a single path to arrive at block 2.1.4. Thus, interfacing actually combines all the implementations that occurred along that one path. For instance, suppose within block 2.1.1 only a partial number of the MADAM behavioral subsets were implemented. If the remainder were then implemented as a whole functional set within block 2.1.2, then 2.1.4 would have to put these implementations together to satisfy the overall requirements for the design. The greatest amount of interfacing would be required if implementations were found for only a partial number of subsets at each of the three levels, with the remainder becoming new circuits during logic synthesis.

Recall that Super-CAD has kept track of the various implementations along the way. Block 2.1.4 must now put them together. Refer to Figure 26.

.1. First, a routine checks to see if the whole design was completed as a single implementation (at either the Functional or Logical level).

.2. If so, the implementation is finalized with whatever bits-and-pieces are required (components such as registers, etc., that may be needed to support the overall requirement).

.3. If a single implementation is not present, then the multiple implementation identifiers are examined to locate the implementations themselves.

.4. Finally, these implementations are interfaced together. This may be a very involved process, as portions of the Functional and

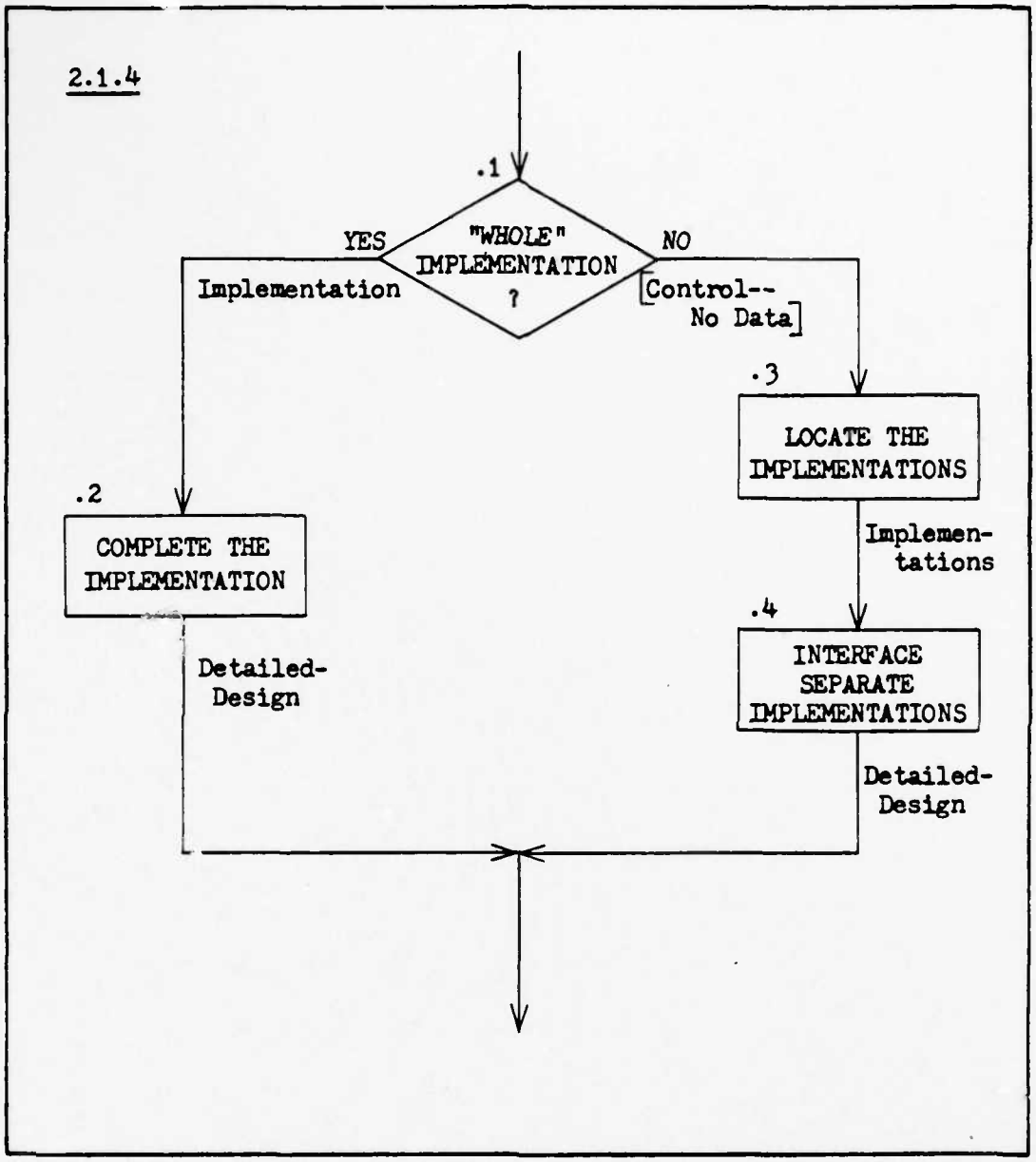



FIG 26. Block 2.1.4

Logical levels are recursively called to complete the interfaces and produce the final design. Because of this complexity, great empnasis will be needed here in future work on the system.

2.1.5. The final block of Figure 15 performs the same functions as 2.1.4.2, for the case when 2.1.1 produces a single implementation from the whole behavioral specification.

2.1 Summary. The central part of the model represented by Figure 15 and the lower level diagrams is a crucial element in the Super-CAD system. Here the single input of a partitioned behavioral specification is molded into a single detailed design output. It is a complex operation with the design effort following but one of many possible paths. As explained earlier in the Realization stage, this is largely hardware-oriented, with software design occurring as a part of it. Software modules are included at each step of implementation until, in the Realization stage, they are broken out, coded, and tested. As Super-CAD grows, this is an area that must be more sharply defined.

2.2. Going back to Figure 14, once block 2.1 produces a detailed design, it is time for Super-CAD to analyze it in block 2.2 (Figure 27).

2.2.1. Figure 27 may appear deceptively simple when, in fact, simulations with the design can be quite complex. Two things help to overcome the complexity. First of all, designing with existing families of IC's means that those blocks have already been tested. The only simulations required are for the interconnections and any new components or IC's added. Secondly, the different levels of abstraction allow simulation of the designs at those levels. It is possible to perform multi-level simulations to test all three levels at once. The fact that different subsets may have been implemented at the different levels

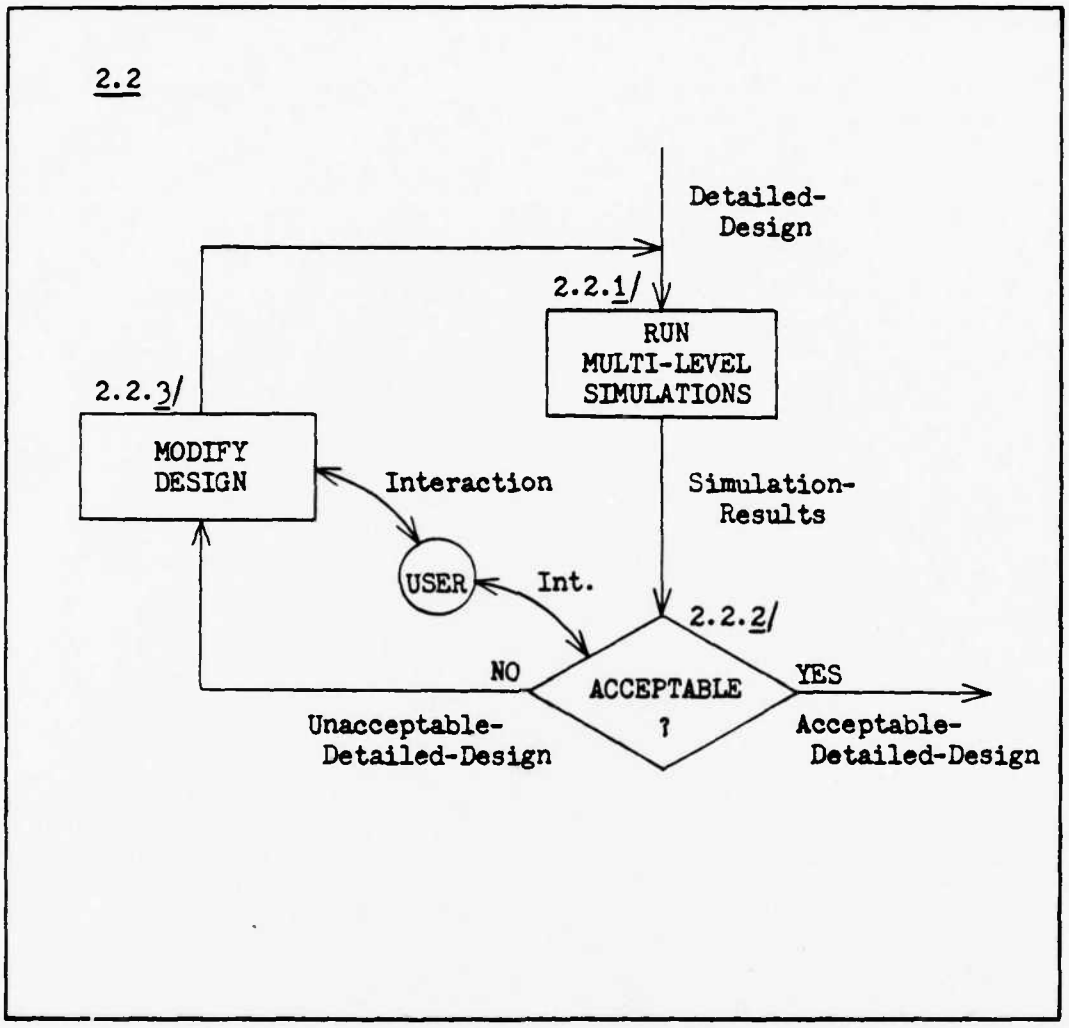



FIG 27. Block 2.2

means that partitioning boundaries already exist, facilitating the simulation.

The Logic 4 program described in Chapter I is an example of a relatively simple simulator using manually formatted input at the Logical level (Ref 26). The biggest limitation in applying Logic 4 to the MADAM design was that it could not handle the whole design at once. Designs of such magnitude will require VLSI-scale tools. But implementing designs at higher levels of abstraction will also be helpful. Thomas describes the benefits of increased speed and less memory requirements using multi-level simulations of VLSI designs (Ref 12:1210). Hachtel also gives a pertinent discussion of VLSI circuit simulation with a "mixed mode" approach that can include (but is not limited to) the different levels of abstraction (Ref 107:1276-1277). Another mixed mode approach is DIANA, for "logic, timing, and circuit simulation" (Ref 108:356).

Specific examples of multi-level simulators include SABLE, a structured approach being developed at Stanford (Refs 76 and 109). Others are MULTI-SIM (Ref 110), MIXS (Ref 111), and an unnamed one described by Agrawal (Ref 112). The SARA system mentioned earlier (Refs 93-96) pursues "multi-level modeling" for a "range of behaviors" (Ref 94:63).

Newton describes several other simulation programs developed at Berkeley, including SPICE, SPLICE, and SAMPLE (Refs 75 and 113). An earlier approach has also applied PERT (Project Evaluation and Review Technique) analysis of critical paths to logic level simulation (Ref 114:152). The SCOAP program discussed in Chapter I, for analyzing the testability of digital circuits (Refs 29 and 30), can also be used at this part of Super-CAD.

Thus, a number of tools are available for use in digital design simulation; Super-CAD may incorporate some, and borrow useful techniques from others. A current project at AFIT will be developing a functional simulation tool (Ref 115) that can be used by Super-CAD. An important area for work is the need for tools to transform the different levels of the specification into test inputs for the multi-level simulations--a process that has usually been done manually by the designer.

2.2.2. The next block in Figure 27, then, interacts with the user to determine if the results of the simulations are acceptable. The results can be displayed on the terminal and can range from messages explaining simulation failures to complete simulation descriptions. Thus, in simulations with the MADAM microprocessor design, a user could quickly determine if it was functioning properly.

2.2.3. If the results are not successful or the user is not satisfied, modifications can be made to the design, interactively. At this point, optimization techniques may come into play. Brayton describes the APLSTAP and DELIGHT systems that work in conjunction with several simulation programs presented earlier (Ref 55:1356-1360). When modifications are complete, the design leaves the analysis block (2.2, Figure 14) and the Implementation stage is finished. The next section explains the Specification stage.

Specification. (Block 1, Figure 7) The final stage for discussion is Specification. It was saved until last because it will be the most difficult for Super-CAD to automate and will require significant advances in design automation before it can be developed completely. This section presents a general treatment of the problem to hopefully point

the direction for some of the future work.

An important perspective on this problem is the application of software development techniques. While Super-CAD is a design system for complete hardware/software packages, the development of the necessary specifications can benefit greatly from work that has already been done in software production.

Essentially, the problem for the Specification stage is to transfrom a relatively simple, unstructured statement of the user's requirements into a well-structured behavioral specification that can be used by the Implementation stage. For the MADAM project, that would mean transforming the requirement to implement the subset of Ada instructions into a specification describing the overall behavior of the instructions. We have already seen the need to employ top-down design and structured design methods (e.g., Refs 57:1302 and 69:618). These principles are especially important during Specification, where the automated design process must get off to a good start. If the specification can be structured to support modularity, the design can be more "technology-independent", and it will be easier to process through the design system (Ref 117:261).

This approach requires an orderly, step-by-step process to define all of the behavior required of the design. Early in the development of Super-CAD, when the rest of the system is getting set up, this part of the design will still be done manually. As mentioned earlier, it may be bypassed entirely by using HDL/RTL inputs to the Implementation stage. But requiring the designers to transform their requirements into one of these languages can be a complicated process. It is one of the ultimate goals of Super-CAD to overcome that necessity. As the Specification

stage evolves and tools are developed, the computer will take over more of these efforts and begin to work interactively with the users. In the progression toward a highly automated stage, the use of artificial intelligence principles should be continually explored.

The user should be able to tell the system that he wants a design that accomplishes a particular set of requirements. Super-CAD will start asking him questions to establish parameters and constraints, make decisions based on the answers, and store the information in the database. It may ask for clarification for some of the decisions or even offer a menu of possible options. The goal is to produce a specification in a form that can be mapped to implementations at the different levels of abstraction.

Partitioning. It was previously explained that the specification must be highly structured and partitionable, to allow the system to assign subsets. Three different approaches might be used in solving the partitioning problem. The first is as described above where the user is responsible. He can input requirements in a hardware description or register-transfer language that intrinsically imposes a high degree of partitioning. The second approach is to make it the responsibility of the high order language used by the overall system. The syntax and semantics of the language may impose partitioning boundaries as the user's input is translated. The third approach is to have the program (Super-CAD) do it. This would be through totally automated algorithms. Presently, there are no known ways to accomplish such automatic partitioning, so it is a significant area for future work. This is a place where artificial intelligence may provide some important answers.

Software Techniques. A number of software engineering tools have

been developed in the past decade, helping to define different phases of the software acquisition process. Some of these tools and techniques can be applied to the definition of a behavioral specification for a complete hardware-software system.

Teichroew and Hershey at the University of Michigan have developed a technique for the "analysis and documentation of requirements and preparation of functional specifications" (Ref 118:41). It is a CAD tool for software development that can offer some useful principles for Super-CAD. Another tool is the Software Engineering Facility (SEF) described earlier (Ref 85). It is really a collection of tools, some of which provide "automated support" for "requirements definition and analysis methods" to assist the user to achieve "maximum results with a minimum of effort" (Ref 85:36-37). An example of a method supported by SEF is SADT* (Structured Analysis and Design Technique). Douglas Ross provides a detailed discussion of requirements definition, structured analysis, and the application of SADT (Refs 119 and 120). The Super-CAD system can adapt these techniques to the specification of overall systems.

DeMarco's text (Ref 71) is an even more detailed description of structured analysis, and Yourdon and Constantine (Ref 70) provide an explanation of structured design. Although neither discussion addresses automation, many of the techniques are significant. Structured design is a high level methodology for the design of the whole software system. It includes structured analysis as one of its tools. Super-CAD should apply the principles of structured design in solving problems; namely, analyze the problem, produce a specification, partition it into modules,

*Trademark of Softech, Inc., Waltham, MA 02154.

and develop the modules as "black boxes" (Ref 70:19,21).

Systems Design Approach. The Manufacturing Technology Division of the Air Force Materials Laboratory at Wright-Patterson Air Force Base employs a Requirements Engineering Methodology (REM) as part of their systems design approach to Integrated Computer-Aided Manufacturing (ICAM). The "systems approach" is a structured one, with a solution to a problem found by decomposing it into smaller problems solved by the "development of subsystems" (Ref 122:2-1). Decomposition of system requirements continues until "each subfunction . . . can be mapped uniquely to one subsystem" (Ref 122:2-26). This is further developed by the IDEF (ICAM Definition) set of modeling methods. The first method, IDEF "zero", "traces its origins to SADT" (Ref 123:B.67). It is a "structured decomposition--the orderly breaking down of a complex subject into its constituent parts." With it, the "method of describing a problem is top-down, modular, hierarchic, and structured" (Ref 123:B.71). The system presents a different perspective that can enhance the Specification stage of Super-CAD through future work. (See also Refs 124 and 128.)

This general discussion of the Specification stage is represented in Figure 28, which expands block 1. First the behavior of the desired system is formalized, and then it is partitioned into a specification.

1.1. Figure 29 shows two steps in identifying the required behavior of the new design. Block 1.1.1 interacts with the user to receive the requirements and clarify them. It draws on the database to present options to the user and sends a set of requirements to the next block. The techniques of structured analysis will be important in this first block. Block 1.1.2 transforms the requirements into a behavioral

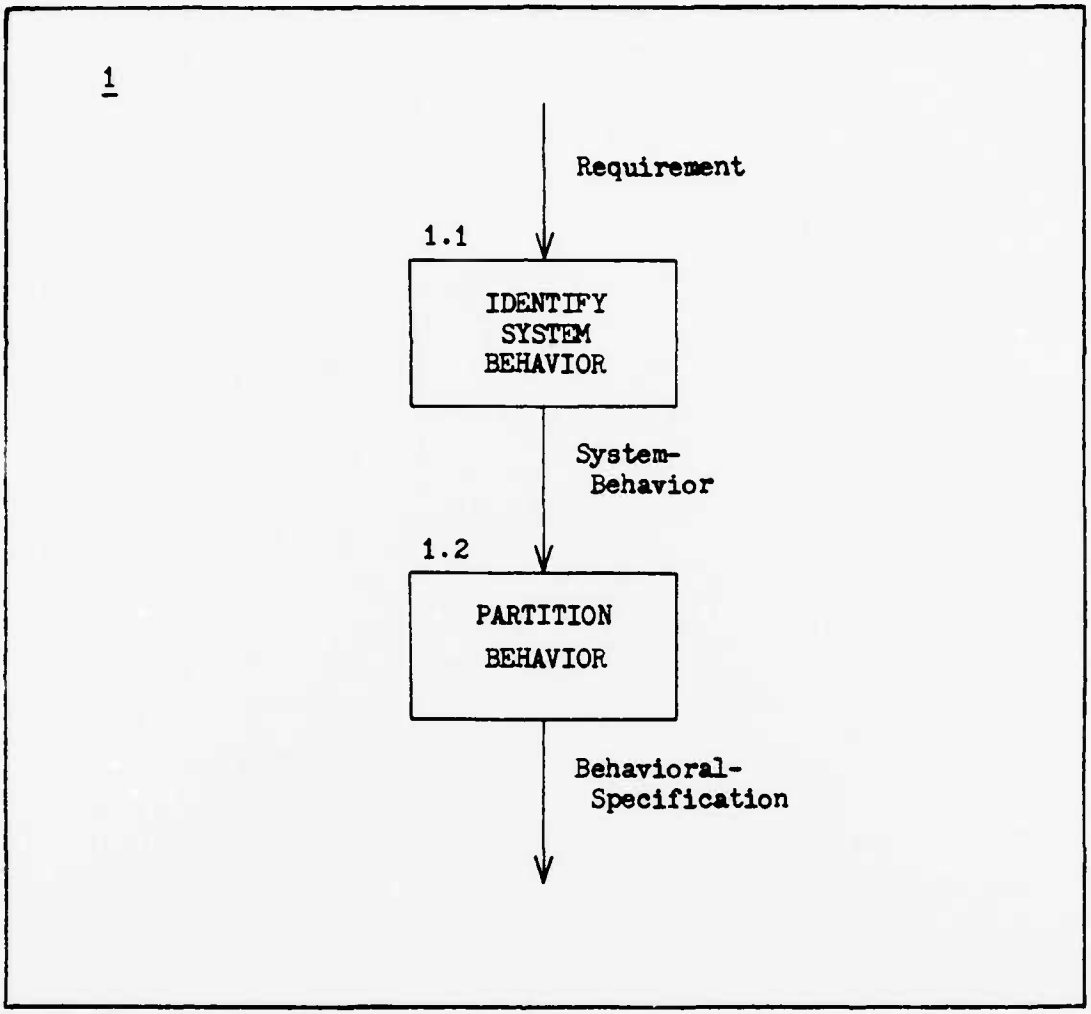



FIG 28. Block 1 - Specification

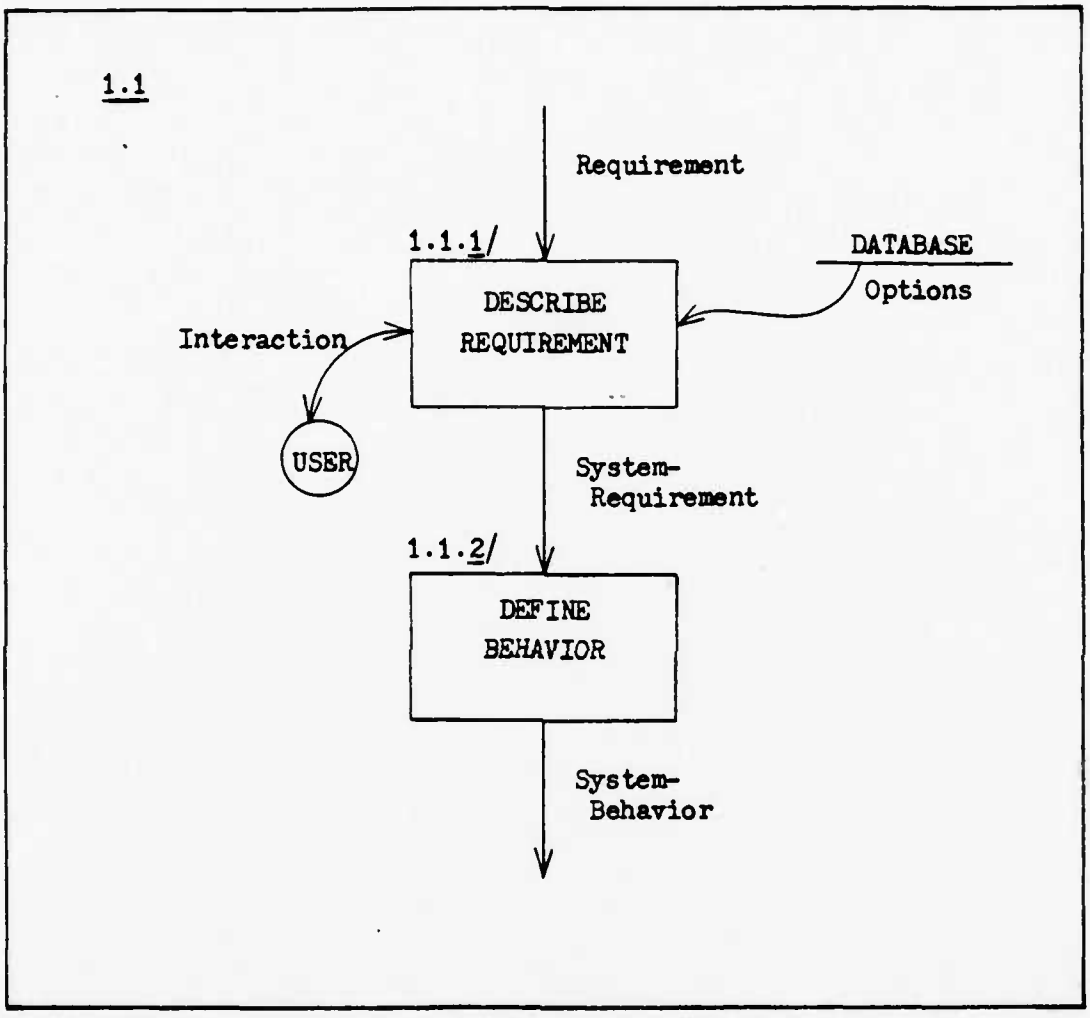



FIG 29. Block 1.1

definition. This is where some of the REM procedures described earlier can be applied.

1.2. The next block (Figure 30) handles the partitioning. First (1.2.1) the behavior is analyzed to define the partition boundaries. Then it is divided into subsets (1.2.2). It is assumed for this model that the subsets are mutually exclusive. As they are divided along the partition lines, portions of the benavior may not fit well into any particular subset. These leftover elements can be combined into one or more additional subsets. For example, it is possible that MADAM subset partitioning might not be strictly according to individual instructions. Since several have overlapping functions, a better partition might be according to the common functions, to maintain the mutually exclusive requirement. In that case, the leftover instruction parts could be combined into other subsets. After this process is complete, the partitioned behavior then goes to the last block (1.2.3) to be formatted into a behavioral specification for the Implementation stage.

While this project has addressed the Specification stage in very general terms, this stage will be a significant part of the Super-CAD system and should be developed fully by later efforts.

Model Conclusion. This, then, is the Super-CAD model. Again, many of the specifics are only suggestions for how to develop it, and future research may define alternate approaches for certain parts. As stated earlier, the appendix reproduces the model diagrams, with the blocks placed in proper numerical order for ease of reference. The next section draws some comparisons of this model with the Carnegie-Mellon project.

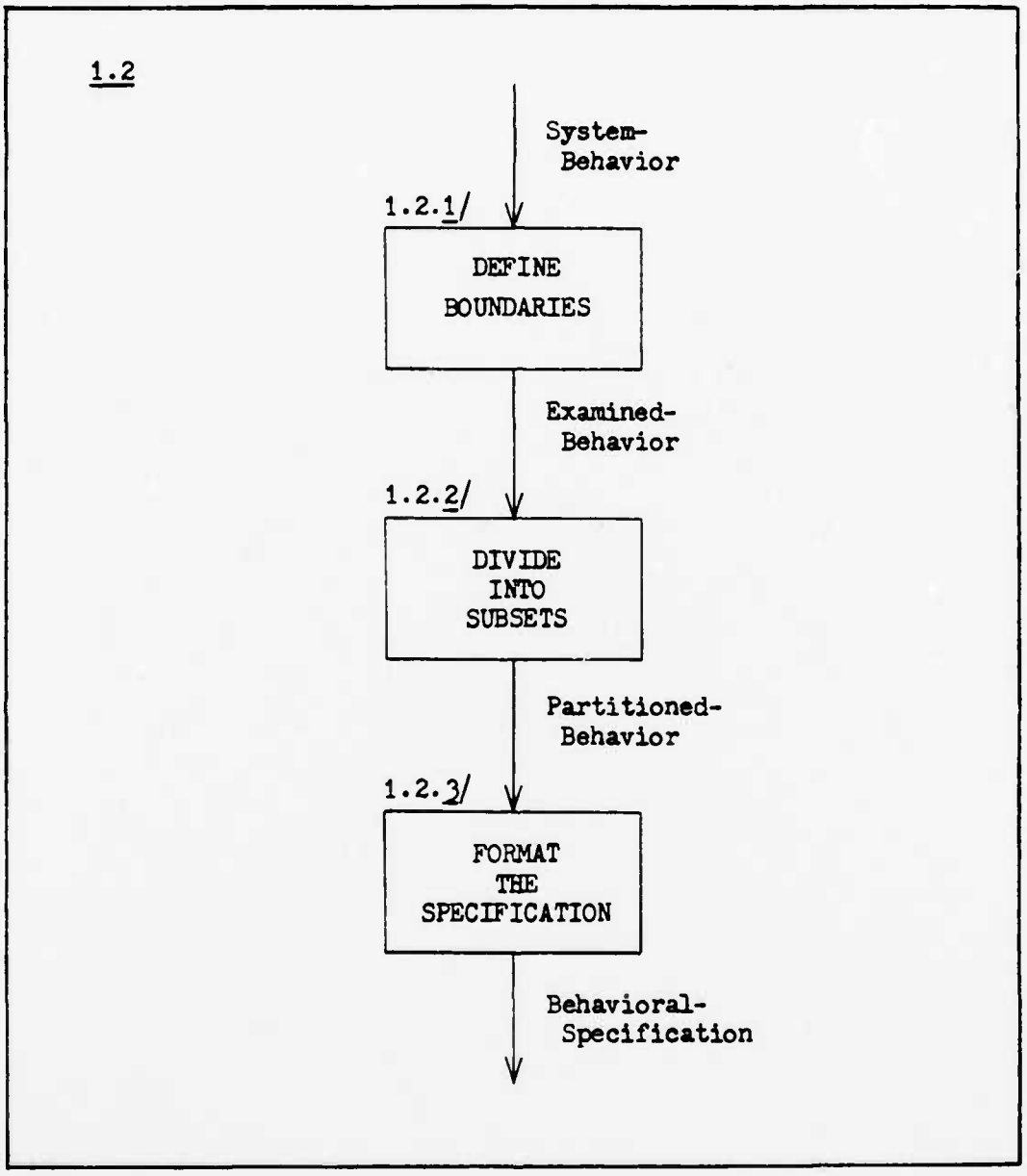



FIG 30. Block 1.2

Comparison (Refs 2, 12, 16, 17, and 36-49)

The CMU-DA project is currently the most ambitious and well-documented effort for integrating the digital design process. Many of the CMU tools have been successfully implemented and are producing encouraging results. A comparison between Super-CAD and CMU-DA can demonstrate similarities and differences to help put the Super-CAD model in perspective.

Both approaches have the same general goal in mind, but with different ways of accomplishing it. That goal is to provide a high-level aid to digital designers in the age of very complex VLSI technologies. The biggest difference between the two is that at the present time Super-CAD is more a philosophy than a system. While several design aids to support Super-CAD are under development, most apply to the Realization stage. With the emphasis of Super-CAD on the Implementation stage, more concrete work needs to be done in that area. CMU-DA, on the other hand, has developed some effective tools for the Implementation stage.

The major emphasis of CMU-DA, in fact, is automating many of the steps in design synthesis and offering "the designer a variety of implementations to choose from" (Ref 44:479). Their input is a high-level "behavioral specification" which really corresponds more closely with Thomas' Functional level. (Some of the authors, in fact, have called it a "functional specification" in some of the earlier papers.) The input is in the ISP language, which means the designer must express his requirements in ISP form.

Super-CAD, as we have seen, attempts to do many of the same things. It proposes to automate the synthesis process and suggests a methodology for solving portions of the problem at successive levels of abstraction.

AD-A118 039 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
SUPER-CAD: AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION.(U)
JUN 82 H S CABLE
UNCLASSIFIED AFIT/GCS/EE/82J-7 NL

2 OF 2
AD
A118 039

END
DATE
FILMED
09-82
DTIC

Where Super-CAD iterates through the different levels, CMU-DA seems to go more directly from the input level of abstraction to the logic realization. CMU-DA essentially ends when it has produced a design that can be sent to layout programs like MP2D. Although Super-CAD could stop at that point, it really includes the layout of IC's and PCB's as part of its integrated effort. At the other end of the spectrum, Super-CAD proposes to automate the Specification stage also. As a result, the designer will not have to understand the intricacies of a design language like ISP to enter his requirements into the system.

Another difference between the two systems is in the type of implementation chosen. Early in the design process, CMU-DA "decides on the specific style of design to be employed (e.g., bit-slice microprocessor, MOS microprocessor, SSI/MSI logic)" (Ref 36:94). Super-CAD basically does not settle on a specific design style, but instead relies on the libraries of various implementations available that might meet the requirements. The type of design becomes more formalized as more implementations are chosen, limiting the additional modules that can be interfaced to them. A factor both systems share, however, is in the flexibility to add new module sets as new designs and technologies become available.

Overall, the two systems share a number of common goals and techniques. Since CMU-DA has many tools at or near completion, the implementation of Super-CAD can benefit immensely from these developments.

Summary

This has been a lengthy treatment of what constitutes the Super-CAD model. In effect, it proposes a method to produce a top-down, structured design that occurs in three stages. First, the user inputs his

requirements and assists the computer in developing a behavioral specification. The computer then proceeds through the three levels of abstraction seeking existing implementations and calling on the user to rule on "close" ones. When potential existing solutions are exhausted, what remains of the specification is made into new circuit designs. Finally, the hardware and software components of the design are realized separately and integrated into a complete system design.

The next chapter explains how this model fits into the design automation research being conducted at AFIT.

## IV. Super-CAD and AFIT

"An in-depth, integrated, and comprehensive design automation program is about to get under way at AFIT." Designed "to meet tne needs of DOD, the Air Force, and AFIT", it can "make a major impact in the future of design automation. And the future of design automation will impact the future of military technology" (Ref 125:10). With these words the current AFIT design automation program began in late 1981. The previous decade had seen some important research in early CAD tools, especially with the CLODS efforts described in Chapter I (Refs 18-24). But now, "as military systems complexities continue to increase", the school is accelerating its efforts to keep up with rapidly changing technology (Ref 125:1).

In AFIT's view, DOD (the Department of Defense) must assure that qualified people are available to meet the challenge of the advancing technology. As an educational and research institution, AFIT can play a significant role by training Air Force personnel and conducting research efforts in design automation. "This capablility is enhanced by faculty members expertly qualified in [DA] to both guide [the] research [and] consult, on a limited basis, for DOD organizations." (Ref 125:1) The Super-CAD project marks the beginning of these efforts to focus on design automation.

### Plan

It is important to examine AFIT's overall plan and see how Super-CAD fits into it. AFIT presently views design automation from three perspectives. The first is the hierarchical view represented by the three levels of abstraction that are at the heart of Super-CAD. The second

view concerns "the various design automation functions that can assist digital engineers" in designing systems (e.g., the difference between hardware and software design). Finally, a third viewpoint highlights, at the physical level, the differences between IC and PCB design, where "there can be distinct differences in complexity." (Ref 125:1-2)

With these perspectives as a basis, AFIT will pursue design automation along several lines. First, it will develop operational tools to support its projects. Of prime concern are the "integrated circuit design courses" which need representative design aids. Also, a number of other "thesis and class projects", both hardware and software, require effective automated tools. (Ref 125:2)

Second is research "on new design automation techniques (and validation of current techniques)". The focus will be on "man-machine interfaces" such as "interactive symbolic languages, hardware description languages," and user-friendly programs. Also important are developing a database to support system integration, using "heirarchical methods in design automation", and exploring "suboptimal techniques" as a means to support "VLSI/VHSIC complexities." (Ref 125:2)

Lastly, faculty "consultation to DOD organizations" is a continuing AFIT requirement. With design automation becoming more crucial, the AFIT research experience will strongly support consultation in this area. (Ref 125:2,4)

The mechanism through which these goals will be accomplished is an integrated system incorporating all AFIT DA software, and organized around a central database system. The "software will reside on a single, dedicated computer system [to be] called the Design Automation Hardware System." Efforts are underway to obtain a suitable system that

can be dedicated solely to the DA research. (Ref 125:4,8)

AFIT will develop software tools that can be directly interfaced to the database. In addition, "other applications software", either "public domain or purchased", "will be interfaced [through] software written at AFIT." The principles of software engineering will be emphasized throughout these developments to promote reliability and ease of testing. (Ref 125:5)

To organize these efforts, the school has developed a five-year plan. The complete plan will not be presented here, but it is divided into several phases that will support the overall sequence of tasks shown in Figure 31. (Ref 125:5) Of course, many of these tasks are interdisciplinary in nature and do not apply solely to design automation. For example, in block 7 artificial intelligence can be tied in to DA, but it is also important in other areas of study such as natural languages or speech- and pattern-recognition. Thus, while some of the projects provide direct support of AFIT DA, others constitute related work with only partial applicability.

The first step is to "specify and define system requirements". Then AFIT will "define and develop the Design Automation Hardware System [and] the data base software capability," while performing "continuing research in three major areas: automated software design, architectural-level design automation, and use of artificial intelligence to solve design automation problems." When the hardware and database have been set up, integrating the DA development process and software tools will begin. (Ref 125:5)

Specific Projects

Super-CAD is the framework within which all of the DA software will

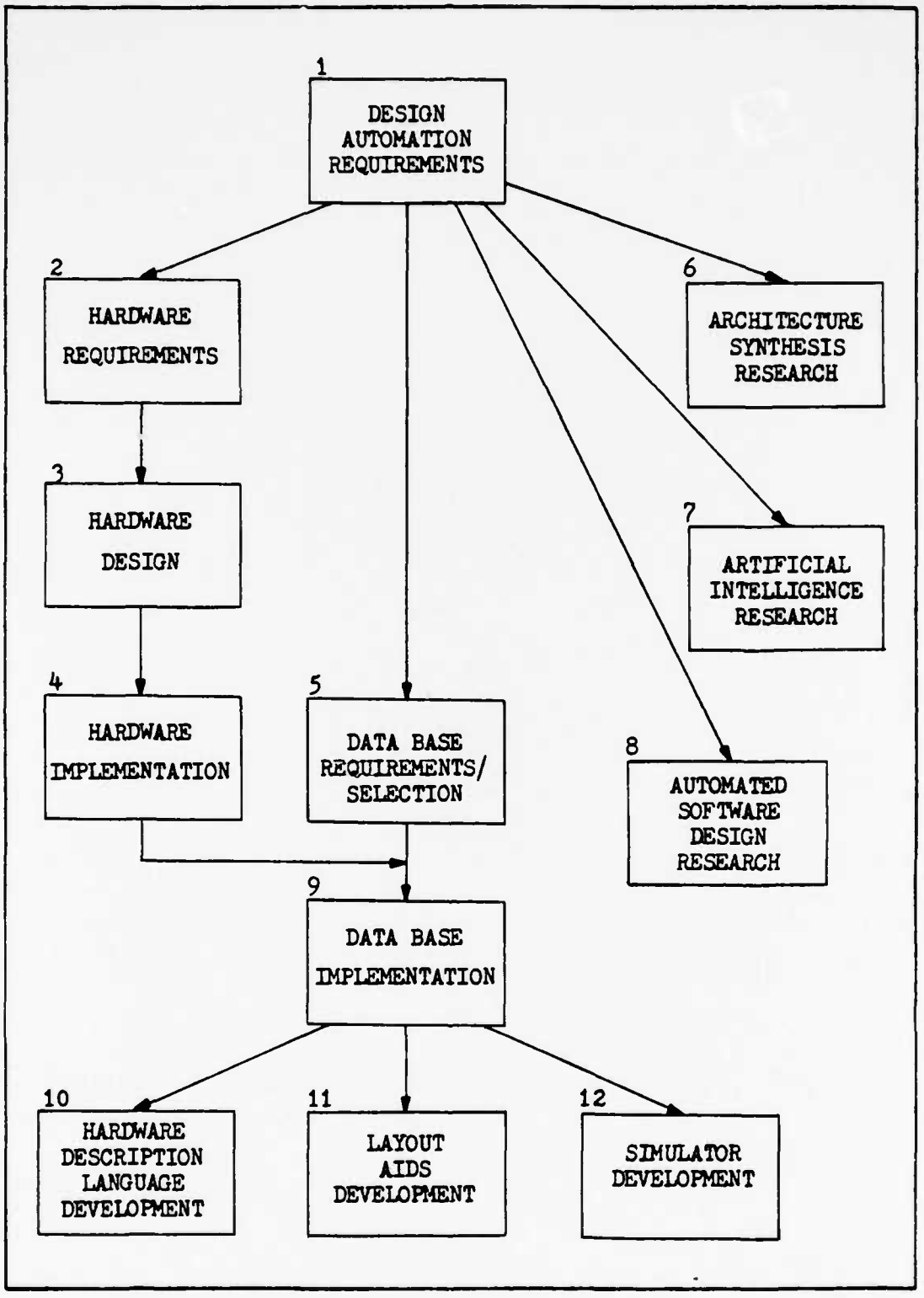



FIG 31. AFIT Design Automation Research and Development Tasks (Ref 125:6)

be integrated. It constitutes a major part of the top block of Figure 31 in defining system requirements. It also contributes to block 6 for "architectural-level design automation" for VLSI/VHSIC "systems in the future" (Ref 125:5).

The other AFIT projects currently in progress, as described in Chapters II and III, fit into various blocks of Figure 31. The database being designed (Ref 65) applies to block 5. The project for automated software development (Ref 121) is the first research effort supporting block 8. Block 11 will benefit from the printed circuit board routing project (Ref 91). Also, the research to apply artificial intelligence to circuit layout (Ref 86) will contribute to both blocks 11 and 7. The functional simulation project (Ref 115) will produce an important tool for block 12. Finally, the design of a graphics work station (Ref 58) will provide a significant input/output capability for the whole system. Other projects to support the system will be getting underway as more students enter the program and become interested in design automation. A new one, in fact, on dynamic testing for integrated circuits has recently begun (Ref 126).

## Recommendations

The various projects just described are a good start for implementing Super-CAD. A significant amount of work remains to be done, however. As we shall see in the next chapter, some important follow-on efforts are required to further define the system and develop some concrete solutions to a number of problem areas. In addition, many of the specific tools under development at AFIT apply to the Realization stage of design. A concerted effort is needed to develop the Implementation stage--to define and implement tools that can aid in the actual design

of a digital system. The Specification stage requires an even greater effort to fill in what is currently just a general proposal. Research can be performed, in conjuction with the Materials Laboratory, to define it better and address some of the problems.

Once AFIT has implemented a few tools in support of Super-CAD, additional projects will be required to interface them with each other in the overall system. A key part of this is the development of the Executive program. Also, while it is already part of the general AFIT plan, specific emphasis should be placed on incorporating existing tools into the system. Valuable CAD tools are available, and many can be useful in Super-CAD, especially in the earlier stages when many holes will be present in the system.

Summary

Super-CAD is an integral part of AFIT's plans for work in design automation. Or, to state it another way, the AFIT DA program--starting with the current five-year plan--will implement Super-CAD. This major undertaking has already begun on several fronts. The serious interest that the school has in DA is perhaps best exemplified by the First Annual Digital System Design Automation Workshop sponsored by AFIT and held in May, 1982. It is planned as a yearly affair and should grow as the DA needs and capabilities of AFIT and the Air Force grow.

## V. Conclusion

The approach to design automation represented by this project has grown out of the need to integrate many available computer-aided design tools and develop new ones for the era of highly complex integrated circuits. In a few short years VLSI and VHSIC chips will contain over a million devices. Digital designers can no longer efficiently create designs for such circuits through manual methods. To be effective, the future design process must rely on integrated sets of tools to assist the designer in all phases.

As reflected in this report, the design process falls naturally into three general stages: Specification, Implementation, and Realization (Figure 1). Also, the design can be represented at three different levels of abstraction: Behavioral, Functional, and Logical (Figure 2). If, during the Implementation stage, designs can be described at the higher levels, fewer details are involved and the design can proceed faster with less computer resources. Then, in the Realization stage, building blocks containing the necessary details can be employed to complete the design.

The evolution of design automation has seen a steady increase in automated tools and some efforts toward supporting design with existing IC's instead of only the design of new ones. A number of examples have been used to show these trends, the most notable of which is the Carnegie-Mellon project. The Super-CAD system proposed here will rely on much of this earlier work and seek to extend it significantly. Ideally, the system will automate all the design stages and produce designs using existing families of circuits at each level of abstraction before creating any new circuits at the lowest level.

Super-CAD will be dual-purpose. The overall, "complete" project will be an integrated system that provides as much design assistance as possible. (It will never really be "complete," requiring continual enhancements.) On the way to the overall configuration, however, it will be a valuable design aid in gathering many effective design tools together in one place. Even in the advanced stages it will still offer the user a variety of individual tools to choose from.

## Recommendations

Since the overall goal of this project has been a high-level definition of the system, a significant amount of additional work is required to advance it toward realization. Many recommendations have already been stated, in one form or another, in the previous chapters. This section summarizes those recommendations and suggests additional areas to be examined in future research.

### The Model.

1. Refine the Super-CAD model as follows:

a. Address the initial assumptions: mutually exclusive subsets, fixed hardware/software split, and combined data/control. The model will be enhanced significantly when it can allow for specification subsets that are not mutually exclusive. Also, the system can be more effective as a design aid if hardware and software development can be separated dynamically at a point more suited to the specific design problem. The third assumption of combining data and control should be examined as well.

b. Examine alternative ways to expand the process blocks. In the development of the model, specific decisions have been made to suggest one approach to the system. Other possibilities can be examined

for suitable alternatives. Also, to better define the system, many of the process blocks in the Implementation stage should be decomposed to levels of greater detail.

c. Develop the Specification stage. Specific steps are needed to define how an input of requirements from a designer can ultimately produce a set of specifications to be passed on to the Implementation stage.

d. Enhance the Implementation stage. Presently, Super-CAD provides for creating new IC's based only on Logical level specifications. This can be expanded to include the same capability at the Functional and Behavioral levels. The "interfacing" block (2.1.4.4) requires further definition to address how to combine the various subset implementations to work together. This step may cause some of the individual implementations to be reworked. Also, a mechanism should be defined for using the problem requirements to help produce test inputs for the simulation step (block 2.2.1). Finally, more specifics on software development should be included in this stage.

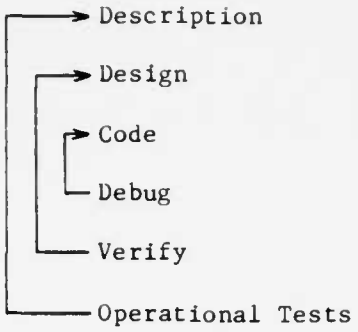
e. Examine the Realization stage for ways to refine current procedures.

2. Add the following capabilities to the model:

a. User inputs to subset partitioning. When the system is unable to produce an adequate number of subset implementations, perhaps the partitioning is at fault. A method is needed to draw the user interactively into revising the subset partitioning based on his knowledge of the requirements. This may produce a more effective partitioning and lead to a successful implementation.

b. Validation. Software Engineering includes the following

steps,




with the arrows showing validation of earlier steps. An important area of research would be to determine if Super-CAD can break out of these constraints and be able to validate the first few steps much earlier, say by the middle levels.

c. Testability. Chapter II discussed the importance of testability in complex digital designs. Further studies are required to assure that Super-CAD incorporates suitable capabilities in this area.

Tools.

1. Define and implement languages for the design stages that are compatible with the model. This report has proposed the use of hardware description and register-transfer languages as a way to input problems directly into the Implementation stage. However, the system should also use a high order language that allows easy user inputs to the Specification stage. Perhaps Ada can be that language. Further study is required, and some interesting questions arise. At how low a level in the model can Ada (or any HOL) be used? Can it be used to describe hardware? For any process block in the model that has an input in Ada, can its output be in Ada? At what level does the answer become "no"? What are the alternatives to Ada, and is a single language enough? Can

applicable data structures be built in Ada? In fact, for any HOL, how will the problem be expressed in the language? Some kind of a structure or specific constructs in the language will be needed so the problem can be described simply and easily. These constructs can be in the form of modules accessed by the input. The interactive nature of the system may mean that Super-CAD questions the user about his problem, perhaps offers a menu of options, and activates modules based on the answers. As stated, this is an area for extensive future work.

2. Investigate the use of methods and techniques like artificial intelligence, group technology, etc., to implement tools for Super-CAD. Artificial intelligence has been mentioned several times in this report. Besides logic synthesis and specification development, it can be applied in many other parts of the model. One project is already underway to apply it to circuit layout.

3. Incorporate existing CAD tools. Early work should emphasize tying together existing CAD programs in support of the Realization stage. This can provide early benefits to users in having the beginnings of an integrated system.

4. Define and implement tools in support of Super-CAD. The Implementation stage has received the most attention in this report, but few automated tools are available to support it. Many must be developed, from relatively straightforward procedures and subroutines to rather complex programs. An example of the latter is the operation to map specification subsets to existing implementations. The other stages, too, require specific tools to be developed, such as subset partitioners, layout routers for VLSI, simulators, etc. An extensive collection of design tools is needed to accomplish the tasks proposed in the

model.

Other.

1. Investigate database technology for support of Super-CAD. The database is a significant part of the system. A current project at AFIT will provide a foundation for development of an effective database system.

2. Examine the relevance of computer architectures to the implementation of Super-CAD. Can the system be hardware/software independent so it can be easily adapted to many computer systems?

3. Define and implement the Executive. A primitive Executive should be developed first, to help interface the initial tools and interact with the user. As more of the system is implemented, the Executive can be updated to accommodate the additions.

4. Examine Initialization, Control and Timing, and Communication in the system. These considerations must be addressed, and perhaps Super-CAD can interact with the user for help in solving them. However, though they can be left to the user in the short-term, they should eventually be handled automatically by the system.

5. Investigate process block interfaces. Work must be done to assure that the outputs of one process can interface directly with the next one. The normal flow in the model supports "synthesis" within a process: to produce a desired output from a given input. However, can the flow be reversed? Can the "analysis" task be performed by having a process examine a realization and determine what problem(s) it can solve? Is the mapping between inputs and outputs always "one-to-one"? If it is "one-to-several" in the forward direction (synthesis), what is it in reverse (analysis)? These are challenging questions for future

research. While Super-CAD proposes to automate the synthesis function, the important question is whether it can automate the analysis function as well.

6. Predict future directions for Super-CAD. While it is difficult to predict what design aids will be required in the future, the Super-CAD system includes the flexibility to adapt as necessary to changing requirements and technologies. Future work with the system should seek to maintain that flexibility.

Summary. Many areas exist for significant future efforts to implement the Super-CAD system. A tentative time-table is suggested below for completion of different development phases. If the system receives support from future AFIT projects, and outside research provides useful tools, the time-table can be met.

Work should begin now, in 1982, to refine the model in the Implementation and Realization stages. Also, efforts can begin to integrate some of the existing CAD programs, through the development of a primitive Executive. This work would extend into the 1983-1984 time-frame. In the meantime, the first generation of support tools being developed at AFIT will be complete in 1983. By then an effort to better define the Specification stage should have begun, and work to describe the additional capabilities can also commence. At the same time, projects to design the next generation of support tools--especially for the Implementation stage--should get underway. Finally, actual development of the database system should also get started.

By 1985, major portions of the model should be clearly defined, work on the Specification stage should be progressing, and many tools supporting the Realization stage should be a part of the system. Gradually,

more tools will come into the Implementation stage, and a few will begin to automate the Specification stage. Between 1985 and 1990 the full Executive can be in operation, with much of the Super-CAD structure filled in with design tools. Many of the problems in automated Specification can be worked out, also. In the decade of the 90's, Super-CAD can be an effective system for performing many digital design functions, while moving toward performing most of them.

## Final Thoughts

The important thing is that, as the Super-CAD concept evolved out of trends in design automation, the system itself will evolve through many phases as more of it is defined and implemented. Gradually it will shift from merely a collection of computer-aided design tools to a complete design automation package supporting all stages of the digital design process. Its strength will lie in its adaptability to changing technologies and changing requirements. It should become a friend of the user by providing individual tools on the one hand and a complete design system on the other. It has the potential to become a significant part of design automation in the future of VLSI/VHSIC, and beyond.

## Bibliography


1. Lattin, William W. et al. "A Methodology for VLSI Chip Design," Lambda Magazine, 2: 34-44 (Second Quarter 1981).

2. Thomas, Donald E. and Daniel P. Siewiorek. "Measuring Designer Performance to Verify Design Automation Systems," IEEE Transactions on Computers, C-30 (1): 48-61 (January 1981).

3. Reitmeyer, Randolph Jr. "CAD for Military Systems, An Essential Link to LSI, VLSI and VHSIC Technology," 18th Design Automation Conference Proceedings. 3-12. Nashville, June 1981.

4. Rosenberg, Lawrence M. "The Evolution of Design Automation to Meet the Challenge of VLSI," 17th Design Automation Conference Proceedings. 3-11. Minneapolis, June 1980.

5. Preiss, Ralph J. "Introduction," Design Automation of Digital Systems, Volume 1: Theory and Techniques, edited by Melvin A. Breuer. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1972.

6. Raymond, Torrance C. "LSI/VLSI Design Automation," Computer, 14 (7): 89-101 (July 1981).

7. Newton, Arthur R. "Computer-Aided Design of VLSI Circuits," Proceedings of the IEEE, 69 (10): 1189-1199 (October 1981).

8. Mead, Carver and Lynn Conway. Introduction to VLSI Systems. Reading, Mass.: Addison-Wesley Publishing Co., 1980.

9. Sangiovanni-Vincentelli, Alberto L. "Guest Editorial," IEEE Transactions on Circuits and Systems, CAS-28 (7): 617 (July 1981).

10. Appleton, Daniel S. "Measure Twice; Cut Once," Datamation, 28: 126-136 (February 1982).

11. Hightower, David. "Can CAD Meet the VLSI Design Problems of the 80's," 16th Design Automation Conference Proceedings. 553. San Diego, June 1979.

12. Thomas, Donald E. "The Automatic Synthesis of Digital Systems," Proceedings of the IEEE, 69 (10): 1200-1211 (October 1981).

13. Siewiorek, Daniel P. and Larry Kwok-Woon Lai. "Testing of Digital Systems," Proceedings of the IEEE, 69 (10): 1321-1333 (October 1981).

14. Wiemann, Warren. "CAD System for VLSI," 16th Design Automation Conference Proceedings. 550. San Diego, June 1979.

15. Rosenberg, Lawrence M. "The Evolution of Design Automation Toward VLSI," Journal of Digital Systems, V: 301-318 (Winter 1981).

16. Kim, Jin H. and Daniel P. Siewiorek. "Issues in IC Implementation of High Level, Abstract Designs," 17th Design Automation Conference Proceedings. 85-91. Minneapolis, June 1980.

17. Barbacci, Mario R. "Instruction Set Processor Specifications for Simulation, Evaluation, and Synthesis," 16th Design Automation Conference Proceedings. 64-72. San Diego, June 1979.

18. Brown, Thomas J. An Automated Digital Design Language for CLODS. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, June 1973. (AD 768 346)

19. Glastetter, Russell A. Automated Design of Digital Integrated Circuit Masks. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1975. (ADA 019 851)

20. Jennings, Lawrence E. The Logic Realization of Automated Design. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, March 1973.

21. Kirk, Fred F. Interactive Graphics Interface for Digital Logic Simulator. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, March 1973. (AD 760 530)

22. Niederhauser, J. Richard. Digital Logic Simulator. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1971. (AD 736 827)

23. Rutledge, James P. Automatic Reduction of Flow Tables. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1970. (AD 880 858)

24. Svisco, Michael J. Computerized Logic-Oriented Design System. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, June 1973. (AD 768 349)

25. Cable, Hobart S. II et al. "MADAM": The Micro Ada Machine Project. Interim and Final Reports of Microprocessor Design project. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, June & September 1981.

26. FATCAT Users Manual, Volume III: Logic 4, Logic Simulation & Fault Analysis Program. Westinghouse Electric Corporation, April 1977.

27. FATCAT Users Manual, Volume VII: IC Layout, CONVRT Program. Westinghouse Electric Corporation, April 1977.

28. Users Guide for the ERADCOM MP2D Program. US Army Electronics Research and Development Command, Electronics Technology & Devices Laboratory.

29. Goldstein, Lawrence H. SCOAP User's Guide: Sandia Controllability/Observability Analysis Program. Albuquerque: Sandia Laboratories, September 1979.

30. Goldstein, Lawrence H. and Evelyn L. Thigpen. "SCOAP: Sandia Controllability/Observability Analysis Program," 17th Design Automation Conference Proceedings. 190-196. Minneapolis, June 1980.

31. Carey, Bernard J. and George F. MacLachlan. "Automated Design Based upon Microprogrammable Bit Slice Microprocessors," Proceedings of the Symposium on Design Automation and Microprocessors. 20-24. Palo Alto, Ca., February 1977.

32. Matelan, M. N. Automating the Design of Dedicated Real Time Control Systems. PhD dissertation. Livermore, Ca.: Lawrence Livermore Laboratory, August 1976.

33. Smith, R. J. and M. N. Matelan. "Practical Considerations In Implementating a Real-Time Controller Design Automation System," Proceedings of the Symposium on Design Automation and Microprocessors. 25-27. Palo Alto, Ca., February 1977.

34. Ross, Alan Albert. Computer Aided Design of Microprocessor-Based Controllers. PhD dissertation. Davis, Ca.: University of California, June 1978.

35. Ross, Alan and Herschel H. Loomis, Jr. "Computer Aided Design of Microprocessor-Based Systems," 15th Design Automation Conference Proceedings. 227-230. Las Vegas, June 1978.

36. Hafer, Louis J. and Alice C. Parker. "Automated Synthesis of Digital Hardware," IEEE Transactions on Computers, C-31 (2): 93-109 (February 1982).

37. Hafer, Louis J. and Alice C. Parker. "Register-Transfer Level Digital Design Automation: The Allocation Process," 15th Design Automation Conference Proceedings. 213-219. Las Vegas, June 1978.

38. Parker, A. et al. "The CMU Design Automation System: An Example of Automated Data Path Design," 16th Design Automation Conference Proceedings. 73-80. San Diego, June 1979.

39. Siewiorek, Dan. "Introducing ISP," Computer, 7 (12): 39-41 (December 1974).

40. Barbacci, Mario R. "Instruction Set Processor Specifications (ISPS): The Notation and Its Applications," IEEE Transactions on Computers, C-30 (1): 24-40 (January 1981).

41. Barbacci, Mario et al. "An architectural research facility--ISP descriptions, simulation, data collection," Proceedings, 1977 National Computer Conference. 161-173. Dallas, June 1977.

42. Thomas, D. E. and G. W. Leive. "A Technology Relative Design System," Proceedings, IEEE International Conference on Circuits and Computers. 1052-1055. Port Chester, N.Y., October 1980.

43. Director, Stephen W. et al. "A Design Methodology and Computer Aids for Digital VLSI Systems," IEEE Transactions on Circuits and Systems, CAS-28 (7): 634-645 (July 1981).

44. Leive, G. W. and D. E. Thomas. "A Technology Relative Logic Synthesis and Module Selection System," 18th Design Automation Conference Proceedings. 479-485. Nashville, June 1981.

45. Tseng, Chia-Jeng and Daniel P. Siewiorek. "The Modeling and Synthesis of Bus Systems," 18th Design Automation Conference Proceedings. 471-478. Nashville, June 1981.

46. Eastman, Charles M. "System Facilities for CAD Databases," 17th Design Automation Conference Proceedings. 50-56. Minneapolis, June 1980.

47. McFarland, Michael C. "On Proving the Correctness of Optimizing Transformations in a Digital Design Automation System," 18th Design Automation Conference Proceedings. 90-97. Nashville, June 1981.

48. Snow, Edward A. et al. "A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations, and Design Tradeoffs," 15th Design Automation Conference Proceedings. 220-226. Las Vegas, June 1978.

49. Thomas, D. E. and D. P. Siewiorek. "Measuring Designer Performance to Verify Design Automation Systems," 14th Design Automation Conference Proceedings. 411-418. New Orleans, June 1977.

50. Matelan, M. N. "Automating the Design of Microprocessor-Based Real Time Control Systems," 13th Design Automation Conference Proceedings. 462-469. San Francisco, June 1976.

51. Breuer, Melvin A. et al. "A Survey of the State of the Art of Design Automation," Computer, 14 (10): 58-75 (October 1981).

52. Daniel, M. E. and C. W. Gwyn. "Hierarchical VLSI Circuit Design," Proceedings, IEEE International Conference on Circuits and Computers. 92-97. Port Chester, N.Y., October 1980.

53. Waxman, R. "VLSI - A Design Challenge," 16th Design Automation Conference Proceedings. 546-547. San Diego, June 1979.

54. Losleben, Paul. "Data Structures, Data Base, and File Management," Digital System Design Automation: Languages, Simulation & Data Base, edited by Melvin A. Breuer. Woodland Hills, Ca.: Computer Science Press, Inc. 1975.

55. Brayton, Robert K. et al. "A Survey of Optimization Techniques for Integrated-Circuit Design," Proceedings of the IEEE, 69 (10): 1334-1362 (October 1981).

56. Rubin, Frank. "A Logic Design Data Entry System," Proceedings, IEEE International Conference on Circuits and Computers. 107-110. Port Chester, N.Y., October 1980.

57. Soukup, Jiri. "Circuit Layout," Proceedings of the IEEE, 69 (10): 1281-1304 (October 1981).

58. Scott, Donna E. "Design Automation Graphics Work Station Design." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

59. Eastman, Charles M. "Database Facilities for Engineering Design," Proceedings of the IEEE, 69, (10): 1249-1263 (October 1981).

60. Ciampi, P. L. et al. "Control and Integration of a CAD Data Base," 13th Design Automation Conference Proceedings. 285-289. San Francisco, June 1976.

61. Ciampi, P. L. and J. D. Nash. "Concepts in CAD Data Base Structures," 13th Design Automation Conference Proceedings. 290-294. San Franscisco, June 1976.

62. Foster, J. C. "The Evolution of an Integration Data Base," 12th Design Automation Conference Proceedings. 394-398. Boston, June 1975.

63. Korenjak, A. J. and A. H. Teger. "An Integrated CAD Data Base System," 12th Design Automation Conference Proceedings. 399-406. Boston, June 1975.

64. Brinton, James B. "CHAS Seeks Title of Global CAD System," Electronics, 54 (3): 100-104 (February 10, 1981).

65. Tebo, Michael. "Design of Conceptual Level Data Base to Support AFIT Design Automation Facility." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

66. Dietmeyer, Donald L. and James R. Duley. "Register Transfer Languages and Their Translation," Digital System Design Automation: Languages, Simulation & Data Base, edited by Melvin A. Breuer. Woodland Hills, Ca.: Computer Science Press, Inc., 1975.

67. Dewey, Al. Digital Eletronics Engineer, Air Force Avionics Laboratory (informal briefing on Hardware Development Language for VHSIC). Wright-Patterson AFB, Ohio, 4 February 1982.

68. Borky, John M. Program Element Monitor for VHSIC at Air Force Systems Command (briefing to students on VHSIC). Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 14 January 1982.

69. Trimberger, Stephen et al. "A Structured Design Methodology and Associated Software Tools," IEEE Transactions on Circuits and Systems, CAS-28 (7): 618-634 (July 1981).

70. Yourdon, Edward and Larry L. Constantine. Structured Design. New York: Yourdon Press, 1978.

71. DeMarco, Tom. Structured Analysis and System Specification. Forward by P. J. Plauger. New York: Yourdon Inc., 1978.

72. Tobias, James R. "LSI/VLSI Building Blocks," Computer, 14 (8): 83-101 (August 1981).

73. Gray, J. P. "Introduction to Silicon Compilation," 16th Design Automation Conference Proceedings. 305-309. San Diego, June 1979.

74. Johannsen, Dave. "Bristle Blocks: A Silicon Compiler," 16th Design Automation Conference Proceedings. 310-313. San Diego, June 1979.

75. Newton, Arthur Richard et al. "Design Aids for VLSI: The Berkeley Perspective," IEEE Transactions on Circuits and Systems, CAS-28 (7): 666-680 (July 1981).

76. Dutton, Robert W. "Stanford Overview in VLSI Research," IEEE Transactions on Circuits and Systems, CAS-28 (7): 654-665 (July 1981).

77. Herrick, Willaim V. and James R. Sims. "A Successful Automated IC Design System," 13th Design Automation Conference Proceedings. 74-78. San Francisco, June 1976.

78. Sansen, Willy et al. "Design Automation Software Towards MOS/VLSI," Proceedings, IEEE International Conference on Circuits and Computers. 98-102. Port Chester, N.Y., October 1980.

79. Groeger, Hans-J. "A New Approach to Structural Partitioning of Computer Logic," 12th Design Automation Conference Proceedings. 378-383. Boston, June 1975.

80. Persky, G. et al. "LTX - A System for the Directed Automatic Design of LSI Circuits," 13th Design Automation Conference Proceedings. 399-407. San Francisco, June 1976.

81. Boehm, Barry W. "Software Engineering," IEEE Transactions on Computers, C-25 (12): 1226-1240 (December 1976).

82. Smith, Robert J. II. "Software Engineering Techniques in Design Automation--A Tutorial," 14th Design Automation Conference Proceedings. 495-507. New Orleans, June 1977.

83. Zelkowitz, Marvin V. "Perspectives on Software Engineering," Computing Surveys, 10 (2): 197-215 (June 1978).

84. Myers, Ware. "The Need for Software Engineering," Computer, 11 (2): 12-24 (February 1978).

85. Irvine, C. A. and John W. Brackett. "Automated Software Engineering Through Structured Data Management," IEEE Transactions on Software Engineering, SE-3 (1): 34-40 (January 1977).

86. Lynch, Robert J. "An Application of Artificial Intelligence to the Circuit Layout Problem." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

87. Welt, Martin J. "NOMAD: A Printed Wiring Board Layout System," 12th Design Automation Conference Proceedings. 152-161. Boston, June 1975.

88. Shupe, Charles F. "Automatic Component Placement in the NOMAD System," 12th Design Automation Conference Proceedings. 162-172. Boston, June 1975.

89. Magnuson, W. G. Jr. "DASLL - An Automatic Printed Circuit Board Layout System," Proceedings, IEEE International Conference on Circuits and Computers. 758-763. Port Chester, N.Y., October 1980.

90. Mori, H. et al. "BRAIN: An Advanced Interactive Layout Design System for Printed Wiring Boards," Proceedings, IEEE International Conference on Circuits and Computers. 754-757. Port Chester, N.Y., October 1980.

91. Chesley, Fred T. "An Automated Design in Printed Circuit Board Routing." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

92. Feller, Albert et al. "Standard Cell Approach for Generating Custom CMOS/SOS Devices Using a Fully Automatic Layout Program," Proceedings, IEEE International Conference on Circuits and Computers. 311-314. Port Chester, N.Y., October 1980.

93. Gardner, Robert J. "State of the Implementation of SARA," Proceedings of the Symposium on Design Automation and Microprocessors. 60-62. Palo Alto, Ca., February 1977.

94. Gardner, Robert J. "Multi-Level Modeling in SARA," Proceedings of the Symposium on Design Automation and Microprocessors. 63-66. Palo Alto, Ca., February 1977.

95. Razouk, Rami R. and Gerald Estrin. "The Graph Model of Behavior Simulator," Proceedings of the Symposium on Design Automation and Microprocessors. 67-76. Palo Alto, Ca., February 1977.

96. Overman, William T. and Gerald Estrin. "Developing a SARA Building Block - The 8080," Proceedings of the Symposium on Design Automation and Microprocessors. 77-86. Palo Alto, Ca., February 1977.

97. McWilliams, Thomas M. and Lawrence C. Widdoes, Jr. "SCALD: Structured Computer-Aided Logic Design," 15th Design Automation Conference Proceedings. 271-277. Las Vegas, June 1978.

98. vanCleemput, W. M. "An Hierarchical Language for the Structural Description of Digital Systems," 14th Design Automation Conference Proceedings. 377-385. New Orleans, June 1977.

99. vanCleemput, W. M. "Computer Hardware Description Languages and Their Applications," 16th Design Automation Conference Proceedings. 554-560. San Diego, June 1979.

100. Chu, Yaohan. "Concepts of a Microcomputer Design Language," 16th Design Automation Conference Proceedings. 45-52. San Diego, June 1979.

101. Friedman, Theodore D. and Sih-Chin Yang. "Methods Used in an Automatic Logic Design Generator (ALERT)," IEEE Transactions on Computers, C-18 (7): 593-614 (July 1969).

102. Zimmerman, G. "The MIMOLA Design System: A Computer Aided Digital Processor Design Method," 16th Design Automation Conference Proceedings. 53-58. San Diego, June 1979.

103. Marwedal, Peter. "The MIMOLA Design System: Detailed Description of the Software System," 16th Design Automation Conference Proceedings. 59-63. San Diego, June 1979.

104. Darringer, John A. et al. "Experiments in Logic Synthesis," Proceedings, IEEE International Conference on Circuits and Computers. 234-237A. Port Chester, N.Y., October 1980.

105. Darringer, John A. and William H. Joyner, Jr. "A New Look at Logic Synthesis," 17th Design Automation Conference Proceedings. 543-549. Minneapolis, June 1980.

106. Máté, Levente L. et al. "CARS: A Computer Aid for Recursive Synthesis," Proceedings, IEEE International Conference on Circuits and Computers. 529-531. Port Chester, N.Y., October 1980.

107. Hachtel, Gary D. and Alberto L. Sangiovanni-Vincentelli. "A Survey of Third-Generation Simulation Techniques," Proceedings of the IEEE, 69 (10): 1264-1280 (October 1981).

108. Reynaert, Ph. et al. "DIANA: A Mixed-Mode Simulator with a Hardware Description Language for Hierarchical Design of VLSI," Proceedings, IEEE International Conference on Circuits and Computers. 356-360. Port Chester, N.Y., October 1980.

109. Hill, Dwight and William vanCleemput. "SABLE: A Tool for Generating Structured, Multi-Level Simulations," 16th Design Automation Conference Proceedings. 272-279. San Diego, June 1979.

110. Chen, Robert C. and James E. Coffman. "Multi-Sim, A Dynamic Multi-Level Simulator, 15th Design Automation Conference Proceedings. 386-391. Las Vegas, June 1978.

111. Sasaki, Tohru et al. "MIXS: A Mixed Level Simulator for Large Digital System Logic Verification," 17th Design Automation Conference Proceedings. 626-633. Minneapolis, June 1980.

112. Agrawal, V. D. et al. "A Mixed-Mode Simulator," 17th Design Automation Conference Proceedings. 618-625. Minneapolis, June 1980.

113. Newton, A. Richard. "Techniques for the Simulation of Large-Scale Integrated Circuits," IEEE Transactions on Circuits and Systems, CAS-26 (9): 741-749 (September 1979).

114. Scheff, Benson H. and Stephen P. Young. "Gate-Level Logic Simulation," Design Automation of Digital Systems, Volume 1: Theory and Techniques, edited by Melvin A. Breuer. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1972.

115. Miles, Max. "The Application of Multiprocessor Architecture Based on the iAPX 432 to Functional Simulation as Part of Digital Design Automation." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

116. Evangelisti, C. J. et al. "Designing with LCD: Language for Computer Design," 14th Design Automation Conference Proceedings. 369-376. New Orleans, June 1977.

117. Bechtolsheim, Andreas. "Interactive Specification of Structured Designs," 15th Design Automation Conference Proceedings. 261-263. Las Vegas, June 1978.

118. Teichroew, Daniel and Ernest A. Hershey, III. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, SE-3 (1): 41-48 (January 1977).

119. Ross, Douglas T. and Kenneth E. Schoman, Jr. "Structured Analysis for Requirements Definition," IEEE Transactions on Software Engineering, SE-3 (1): 6-15 (January 1977).

120. Ross Douglas T. "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transctions on Software Engineering, SE-3 (1): 16-34 (January 1977).

121. Hadfield, Steven M. "An Automated Software Development Environment." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

122. Alford, M. W. et al. Formal Decomposition Applied to Axiomatic Requirements Engineering. Technical Report. Huntsville, Ala.: TRW Defense and Space Systems Group, December 1979.

123. ICAM Industry Days, Volume 2, Technical Summary. St. Louis, September 1980.

124. Sixth Annual ICAM Industry Days Proceedings. New Orleans, January 1982.

125. Carter, Harold W. "A Plan for Digital Systems Design Automation at the Air Force Institute of Technology," planning document, Department of Electrical Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, November 1981.

126. Young, David. "A Dynamic Test Model for Integrated Circuits Based on Static Methods." Unpublished MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, 1982.

127. Ludewig, Jochen. "Computer-Aided Specification of Process Control Systems," Computer, 15 (5): 12-20 (May 1982).

128. Computer, 15 (5): 10-59 (May 1982). Special Issue on Application-Oriented Specifications for Software Systems.

129. Garlington, Alan R. Preliminary Design and Implementation of an Ada Pseudo-Machine. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, March 1981.

130. Hayes, John P. Computer Architecture and Organization. New York: McGraw-Hill Book Co., 1978.

131. Sippl, Charles J. and Roger J. Sippl. Computer Dictionary and Handbook. Indianapolis: Howard W. Sams & Co., Inc., 1980.

132. Korn, Granino A. Microprocessors and Small Digital Computer Systems for Engineers and Scientists. New York: McGraw-Hill Book Co., 1977.

General References and Recommended Reading

133. Breuer, Melvin A., ed. Design Automation of Digital Systems, Volume 1: Theory and Techniques. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1972.

134. Breuer, Melvin A., ed. Digital System Design Automation: Languages, Simulation & Data Base. Woodland Hills, Ca.: Computer Science Press, Inc., 1975.

135. Computer, 7 (12): 18-51 (December 1974). Special Issue on Hardware Description Languages.

136. Dutton, Robert W. and Stephen E. Hansen. "Process Modeling of Integrated Circuit Device Technology," Proceedings of the IEEE, 69 (10): 1305-1320 (October 1981).

137. IEEE Transactions on Software Engineering, SE-3 (1): 6-84 (January 1977). Special Issue on Requirement Analysis.

138. Journal of Digital Systems, V: 299-451 (Winter 1981). Special Issue on Design Automation.

139. Kawano, Ietoshi et al. "The Design of a Data Base Organization for an Electronic Equipment DA System," 15th Design Automation Conference Proceedings. 167-175. Las Vegas, June 1978.

140. Mano, M. Morris. Digital Logic and Computer Design. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1979.

141. Miller, G. A. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," Psychology Review, 63 (3): 81-97 (March 1956).

142. Proceedings, IEEE International Conference on Circuits and Computers. Port Chester, N.Y., October 1980. Numerous papers on VLSI and Design Automation.

143. Proceedings of the Symposium on Design Automation and Microprocessors. Palo Alto, Ca., February 1977.

144. Proceedings of the IEEE, 69 (10): 1187-1362 (October 1981). Special Issue on Computer-Aided Design.

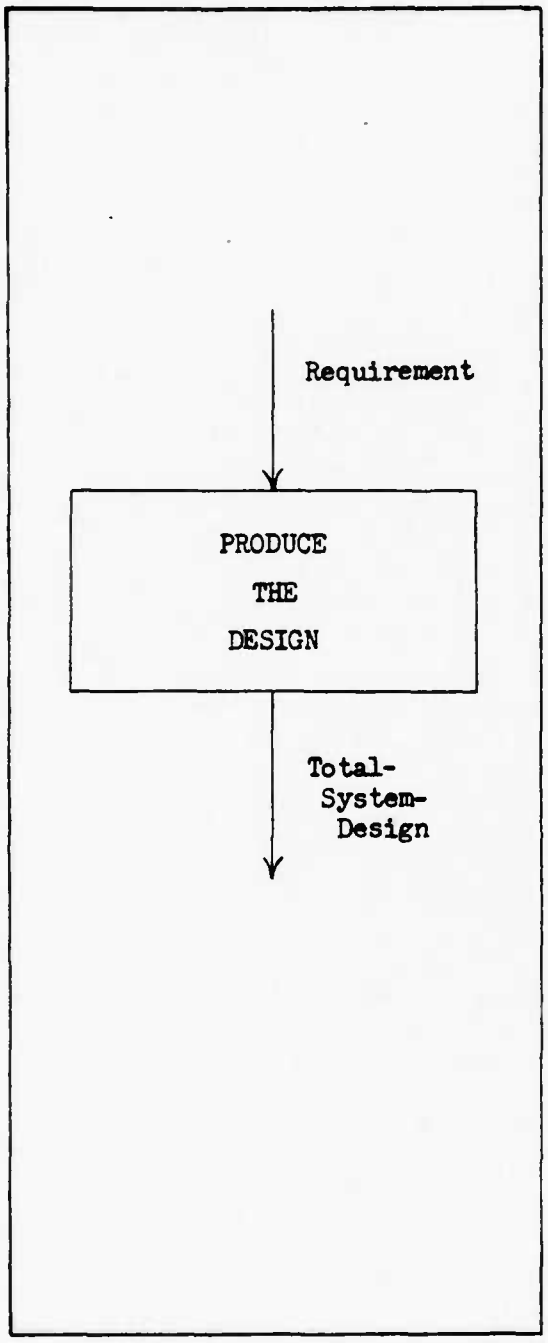
145. Siewiorek, Dan. "Introducing PMS," Computer, 7 (12): 42-44 (December 1974).

146. 12th to 18th Design Automation Conference Proceedings. 1975-1981. (Called Design Automation Workshops prior to that.) Many pertinent papers.

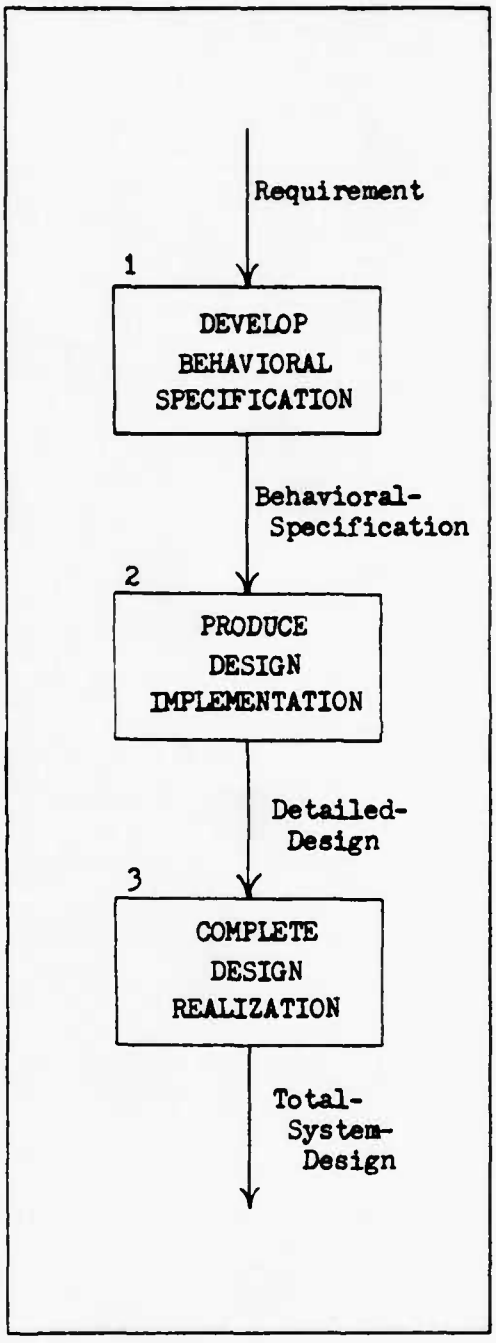
147. Westerberg, Arthur W. "Computer-Aided Design Tools in Chemical Engineering Process Design," Proceedings of the IEEE, 69 (10): 1232-1239 (October 1981).

# Appendix

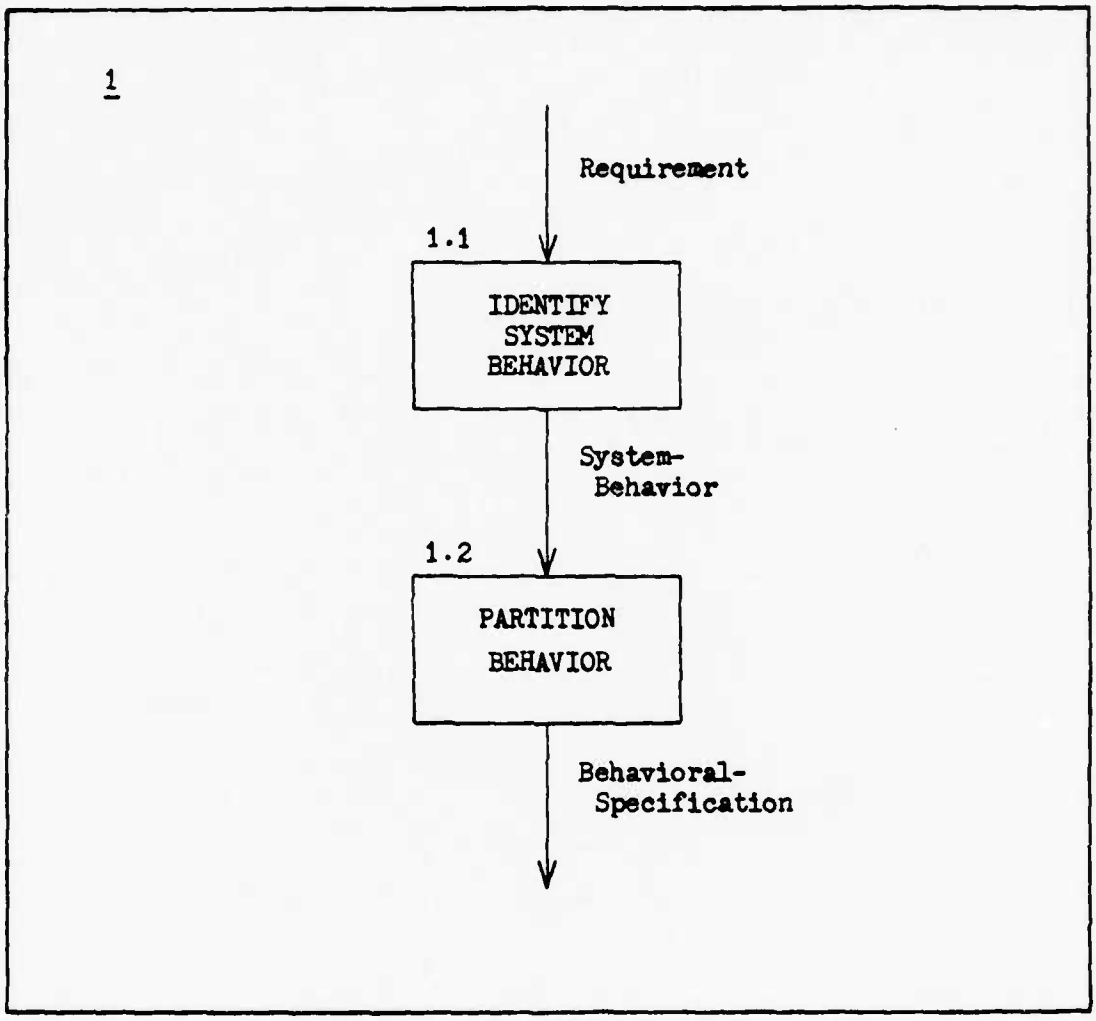
## COMPENDIUM OF MODEL DIAGAMS



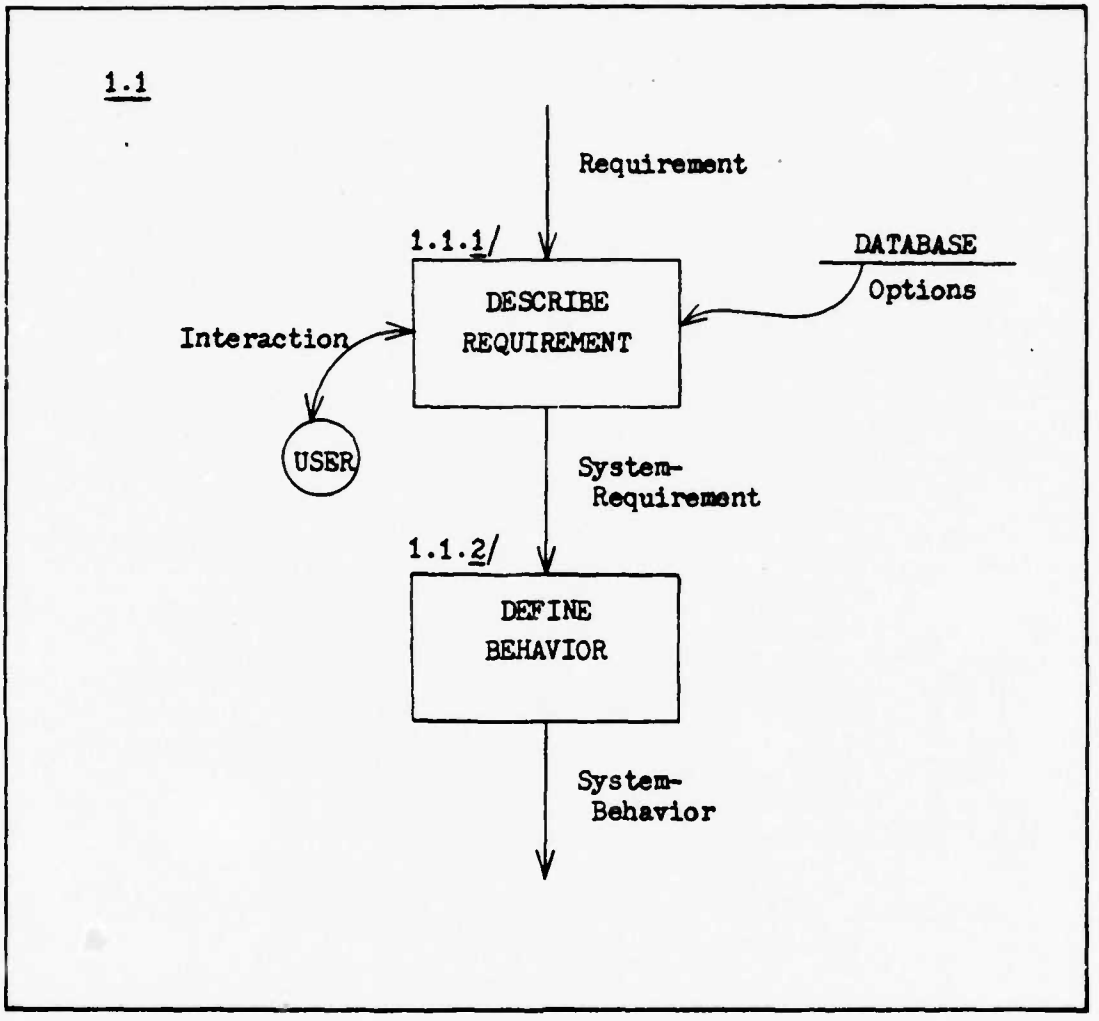

Overall Super-CAD Process



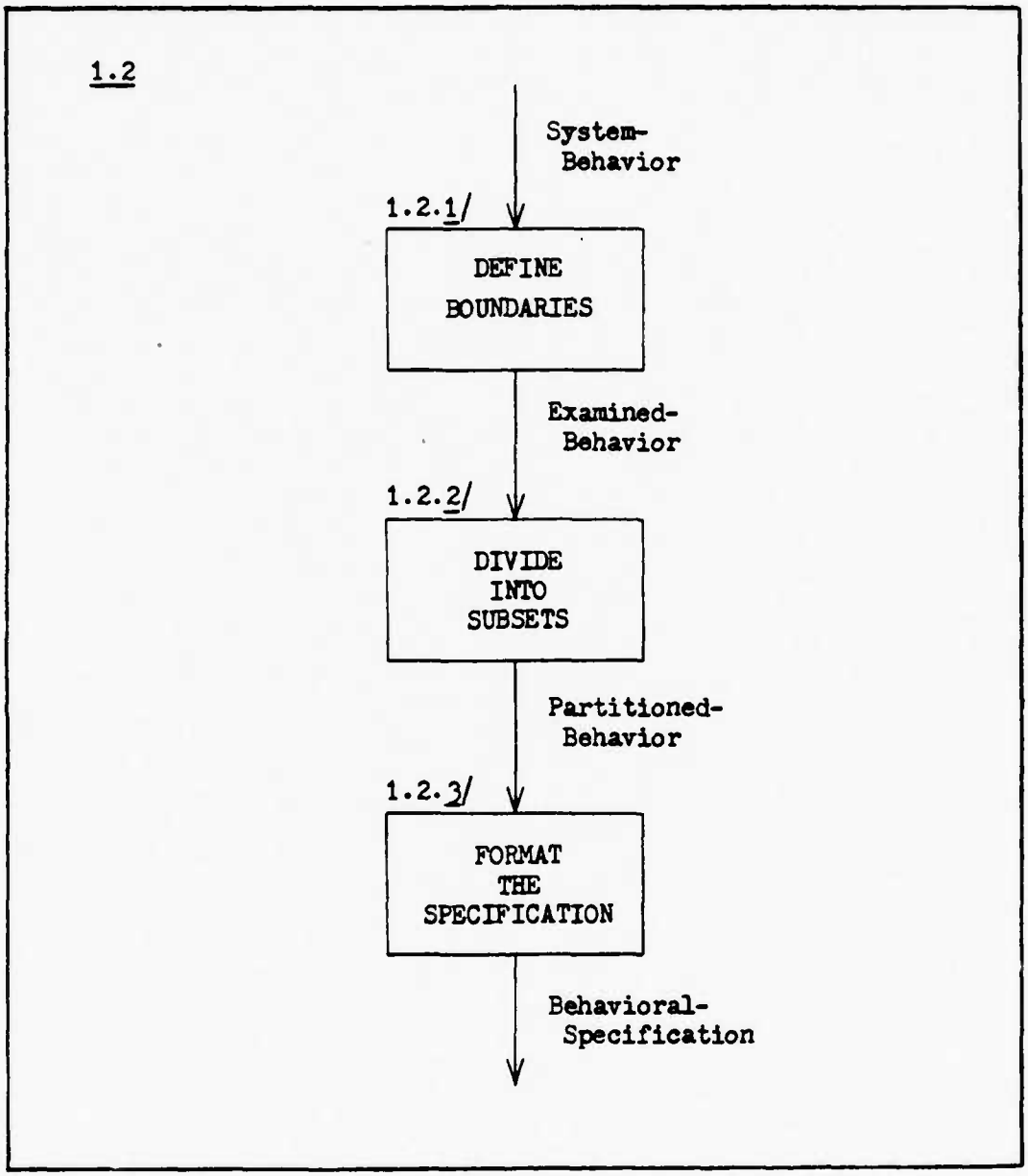

Main Design Stages


1
Requirement
1.1
IDENTIFY SYSTEM BEHAVIOR
System-Behavior
1.2
PARTITION BEHAVIOR
Behavioral-Specification

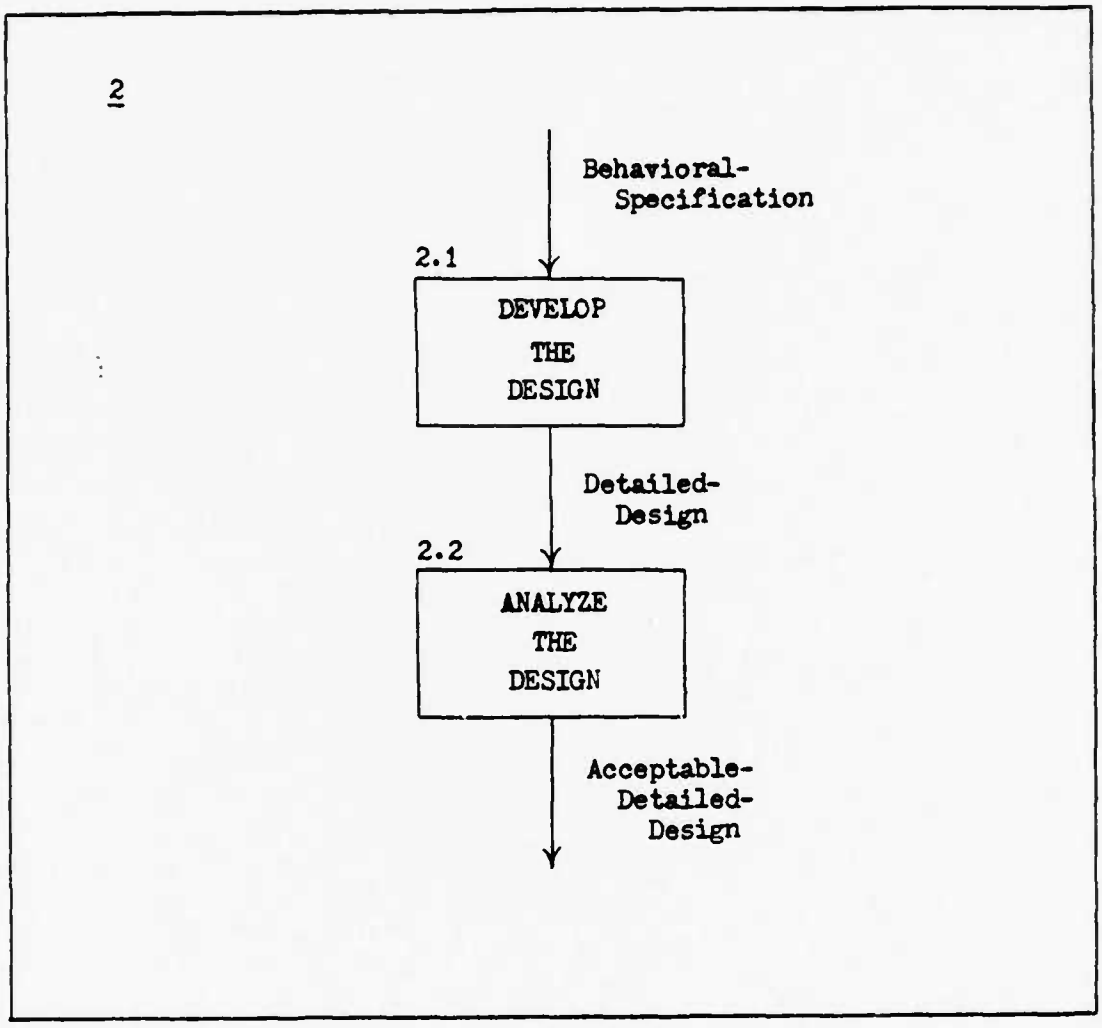
Block 1


1.1
Requirement
1.1.1/
DESCRIBE
REQUIREMENT
DATABASE
Options
Interaction
USER
System-
Requirement
1.1.2/
DEFINE
BEHAVIOR
System-
Behavior

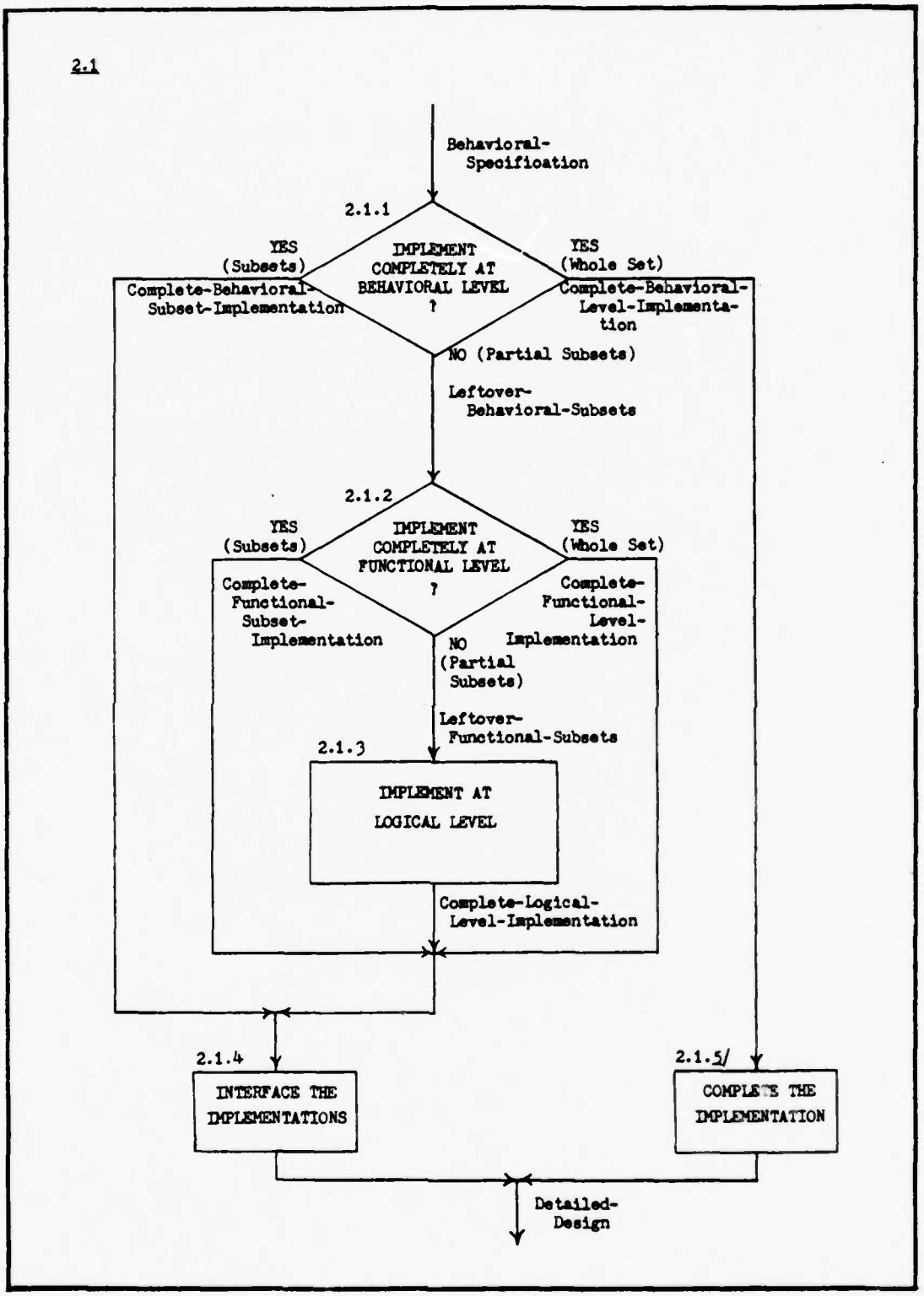
Block 1.1


1.2
System-
Behavior
1.2.1/
DEFINE
BOUNDARIES
Examined-
Behavior
1.2.2/
DIVIDE
INTO
SUBSETS
Partitioned-
Behavior
1.2.3/
FORMAT
THE
SPECIFICATION
Behavioral-
Specification

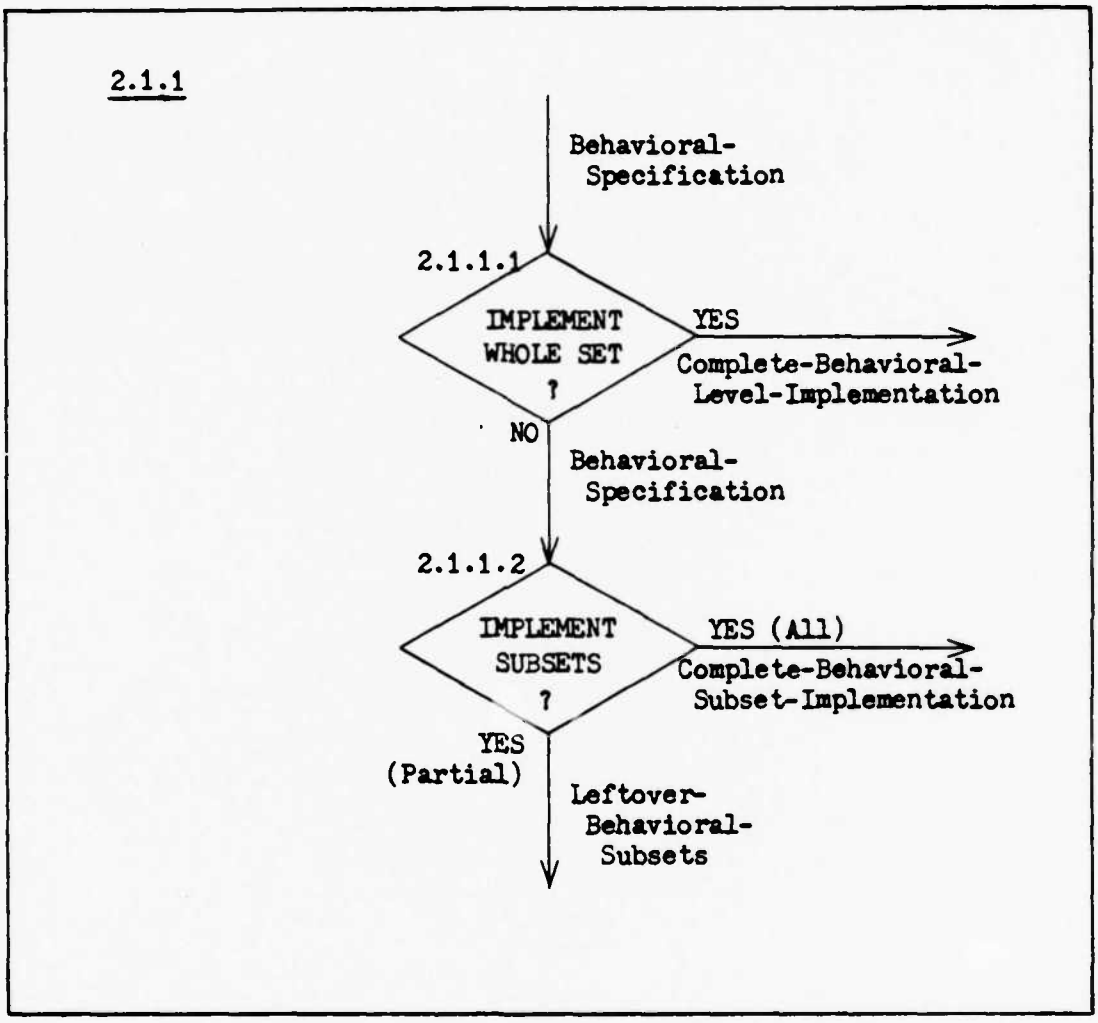
Block 1.2


2
Behavioral-
Specification
2.1
DEVELOP
THE
DESIGN
Detailed-
Design
2.2
ANALYZE
THE
DESIGN
Acceptable-
Detailed-
Design

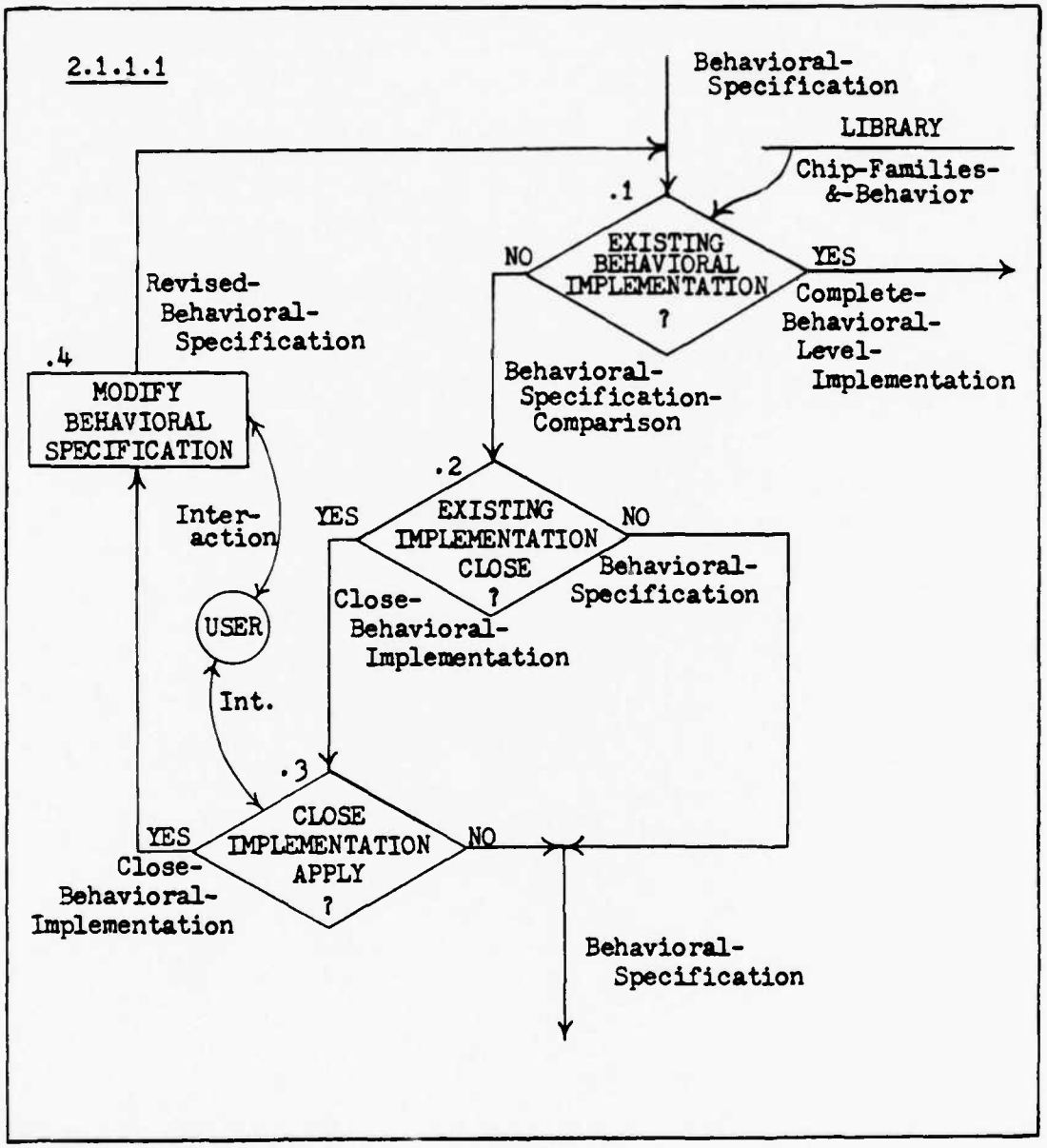
Block 2


2.1
Behavioral-Specification
2.1.1
IMPLEMENT COMPLETELY AT BEHAVIORAL LEVEL ?
YES (Subsets)
Complete-Behavioral-Subset-Implementation
YES (Whole Set)
Complete-Behavioral-Level-Implementation
NO (Partial Subsets)
Leftover-Behavioral-Subsets
2.1.2
IMPLEMENT COMPLETELY AT FUNCTIONAL LEVEL ?
YES (Subsets)
Complete-Functional-Subset-Implementation
YES (Whole Set)
Complete-Functional-Level-Implementation
NO (Partial Subsets)
Leftover-Functional-Subsets
2.1.3
IMPLEMENT AT LOGICAL LEVEL
Complete-Logical-Level-Implementation
2.1.4
INTERFACE THE IMPLEMENTATIONS
2.1.5/
COMPLETE THE IMPLEMENTATION
Detailed-Design

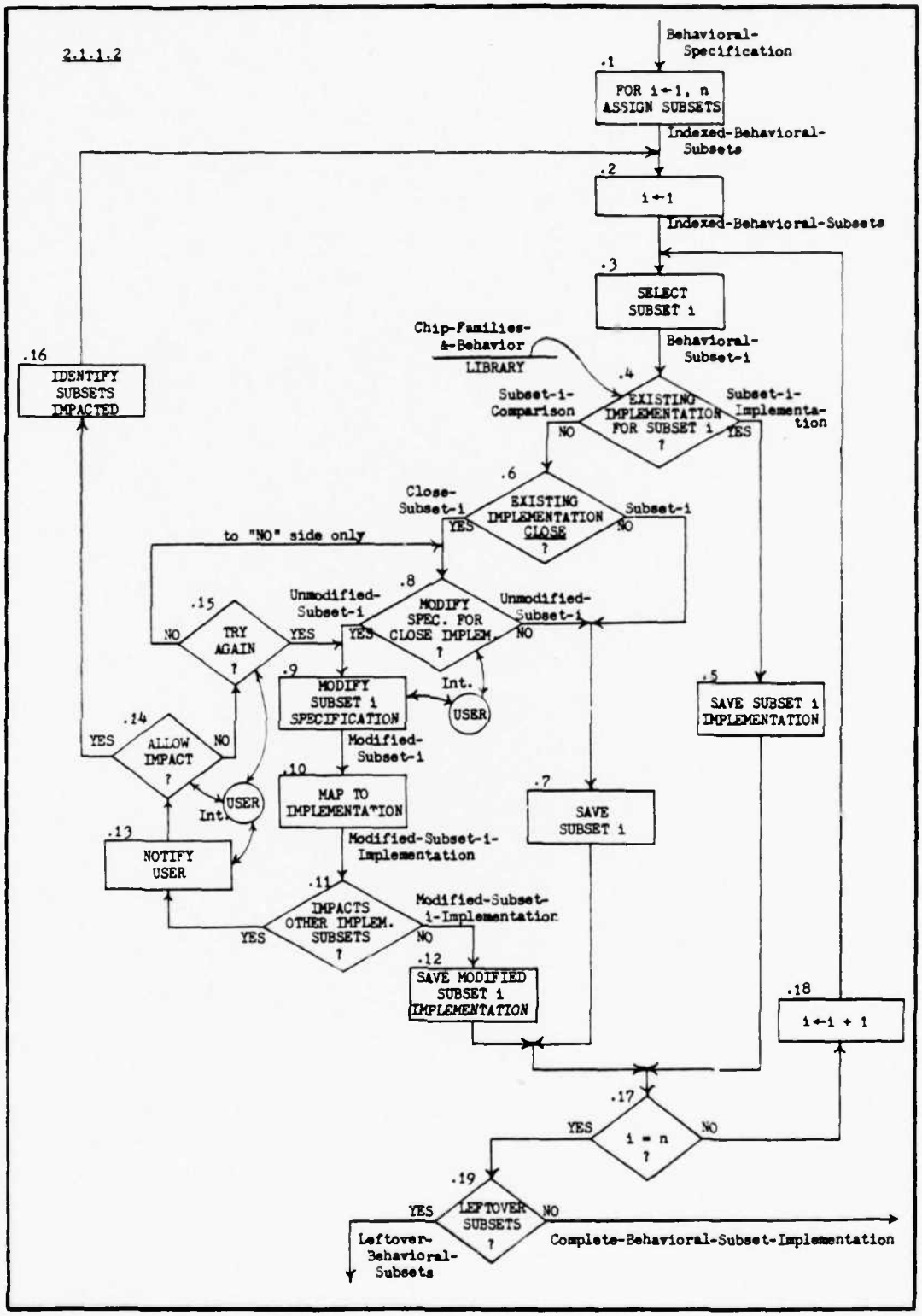
Block 2.1


2.1.1
Behavioral-
Specification
2.1.1.1
IMPLEMENT
WHOLE SET
?
YES
Complete-Behavioral-
Level-Implementation
NO
Behavioral-
Specification
2.1.1.2
IMPLEMENT
SUBSETS
?
YES (All)
Complete-Behavioral-
Subset-Implementation
YES
(Partial)
Leftover-
Behavioral-
Subsets

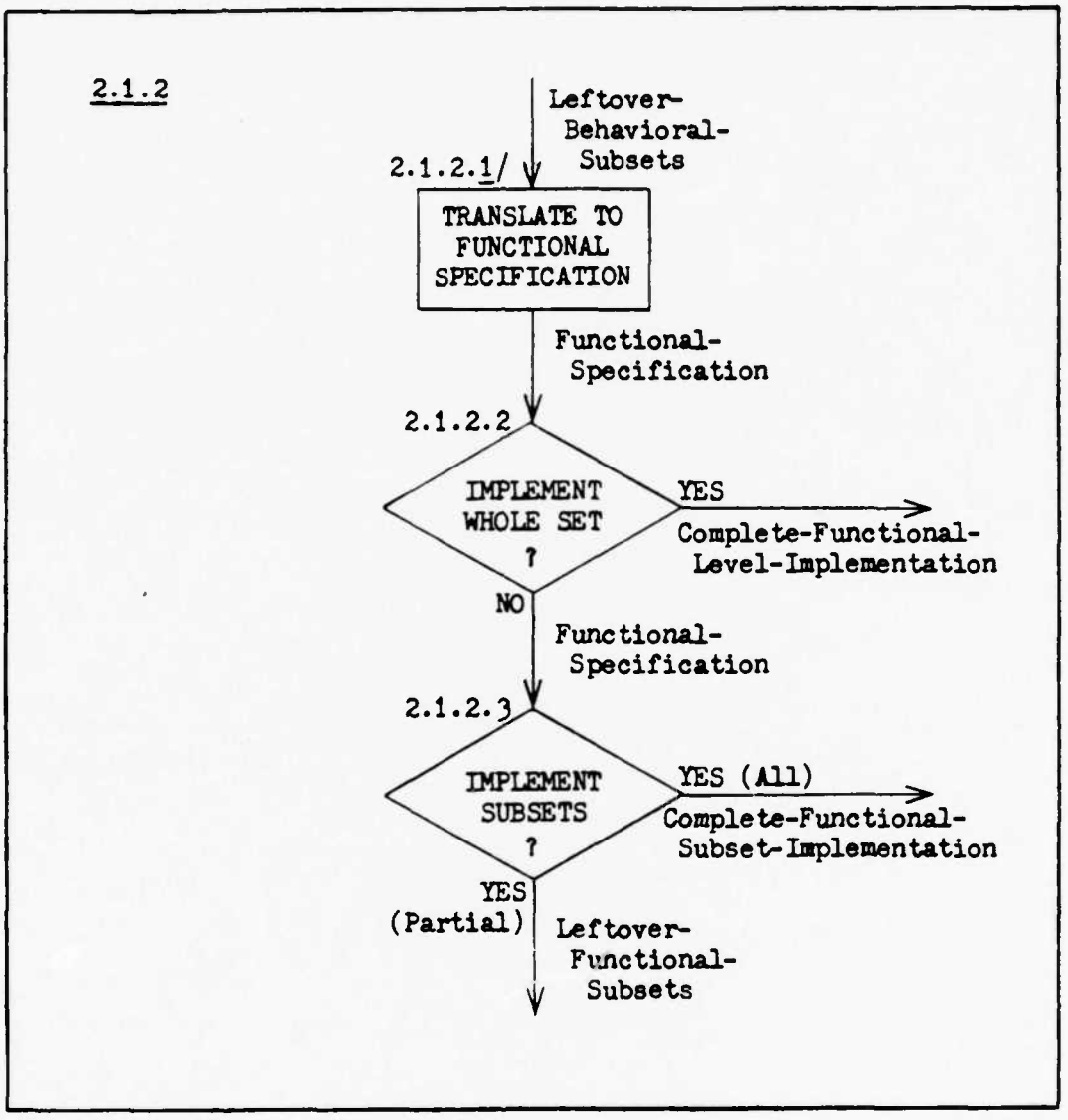
Block 2.1.1


2.1.1.1
Behavioral-
Specification
LIBRARY
Chip-Families-
&-Behavior
.1
EXISTING
BEHAVIORAL
IMPLEMENTATION
?
NO
YES
Complete-
Behavioral-
Level-
Implementation
Revised-
Behavioral-
Specification
.4
MODIFY
BEHAVIORAL
SPECIFICATION
Behavioral-
Specification-
Comparison
.2
EXISTING
IMPLEMENTATION
CLOSE
?
YES
NO
Inter-
action
Behavioral-
Specification
Close-
Behavioral-
Implementation
USER
Int.
.3
CLOSE
IMPLEMENTATION
APPLY
?
YES
NO
Close-
Behavioral-
Implementation
Behavioral-
Specification

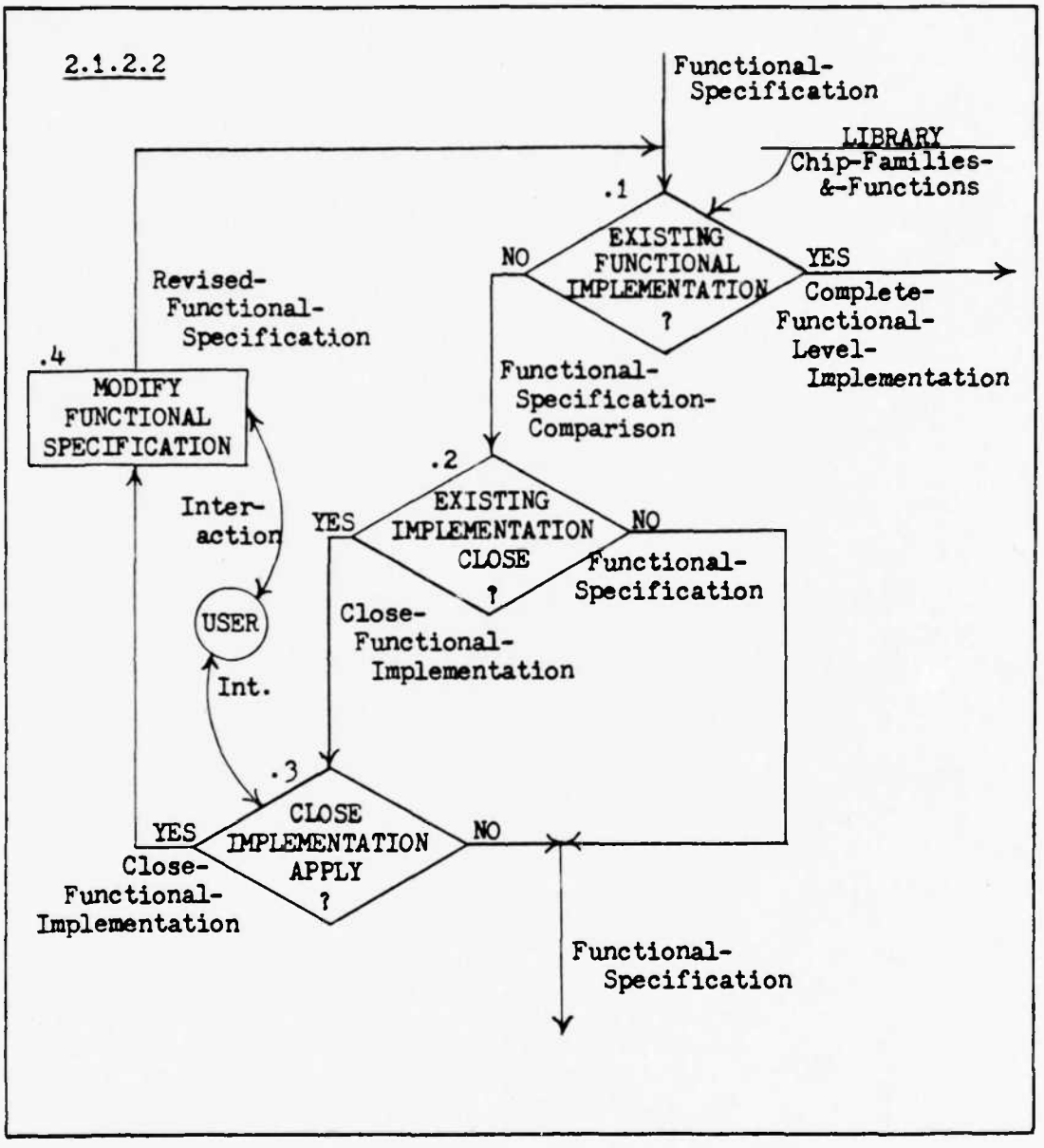
Block 2.1.1.1


2.1.1.2
Behavioral-Specification
.1
FOR i←1, n ASSIGN SUBSETS
Indexed-Behavioral-Subsets
.2
i←1
Indexed-Behavioral-Subsets
.3
SELECT SUBSET i
Chip-Families-&-Behavior LIBRARY
Behavioral-Subset-i
.4
EXISTING IMPLEMENTATION FOR SUBSET i ?
Subset-i-Comparison
NO
YES
Subset-i-Implementation
.6
EXISTING IMPLEMENTATION CLOSE ?
Close-Subset-i
YES
NO
Subset-i
to "NO" side only
.8
MODIFY SPEC. FOR CLOSE IMPLEM. ?
Unmodified-Subset-i
YES
NO
Unmodified-Subset-i
.15
TRY AGAIN ?
NO
YES
.9
MODIFY SUBSET i SPECIFICATION
Int.
USER
.5
SAVE SUBSET i IMPLEMENTATION
.14
ALLOW IMPACT ?
YES
NO
Modified-Subset-i
.10
MAP TO IMPLEMENTATION
USER
Int.
.7
SAVE SUBSET i
.13
NOTIFY USER
Modified-Subset-i-Implementation
.11
IMPACTS OTHER IMPLEM. SUBSETS ?
YES
NO
Modified-Subset-i-Implementation
.12
SAVE MODIFIED SUBSET i IMPLEMENTATION
.16
IDENTIFY SUBSETS IMPACTED
.18
i←i + 1
.17
i = n ?
YES
NO
.19
LEFTOVER SUBSETS ?
YES
NO
Leftover-Behavioral-Subsets
Complete-Behavioral-Subset-Implementation

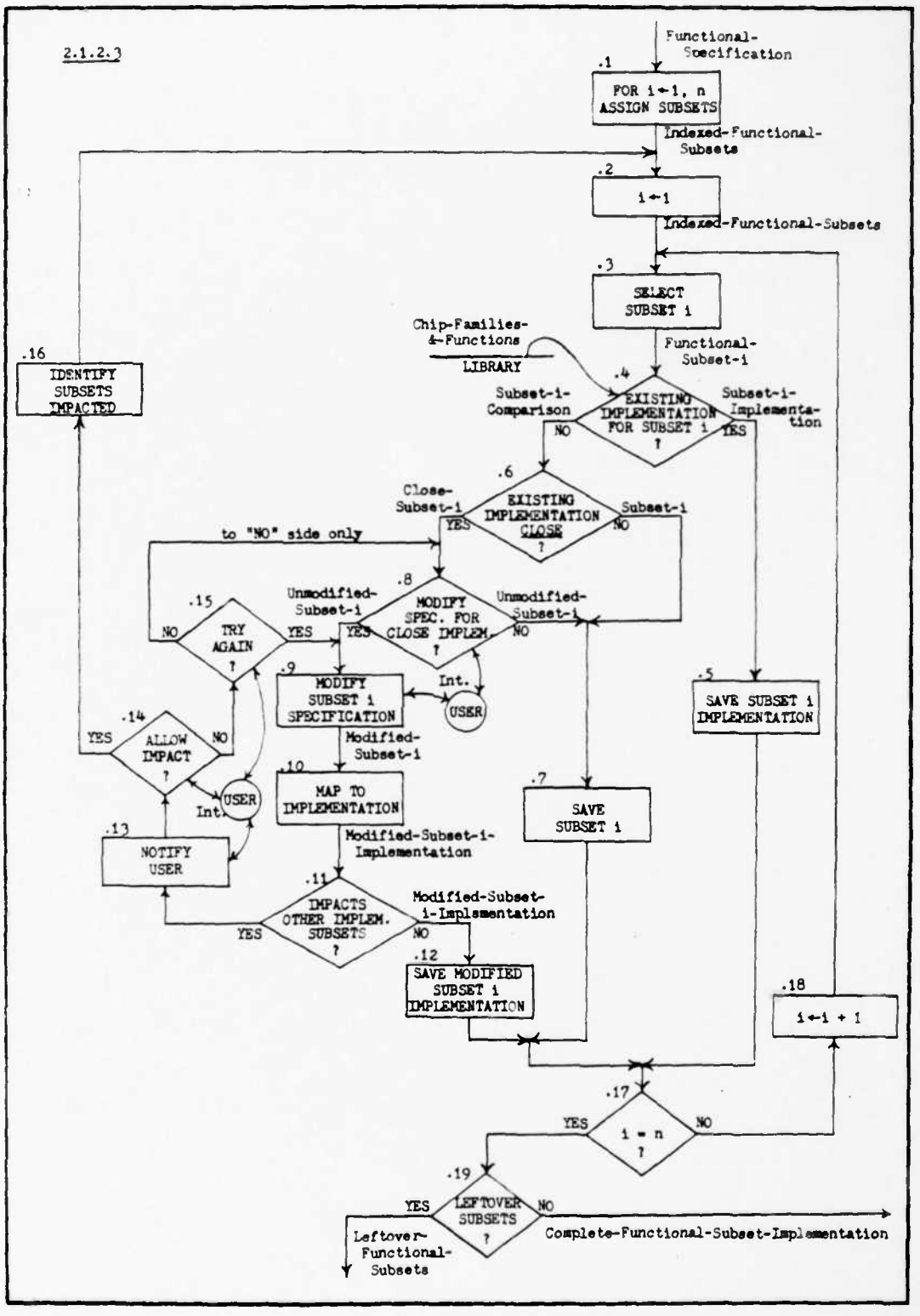
Block 2.1.1.2


2.1.2
Leftover-
Behavioral-
Subsets
2.1.2.1/
TRANSLATE TO
FUNCTIONAL
SPECIFICATION
Functional-
Specification
2.1.2.2
IMPLEMENT
WHOLE SET
?
YES
Complete-Functional-
Level-Implementation
NO
Functional-
Specification
2.1.2.3
IMPLEMENT
SUBSETS
?
YES (All)
Complete-Functional-
Subset-Implementation
YES
(Partial)
Leftover-
Functional-
Subsets

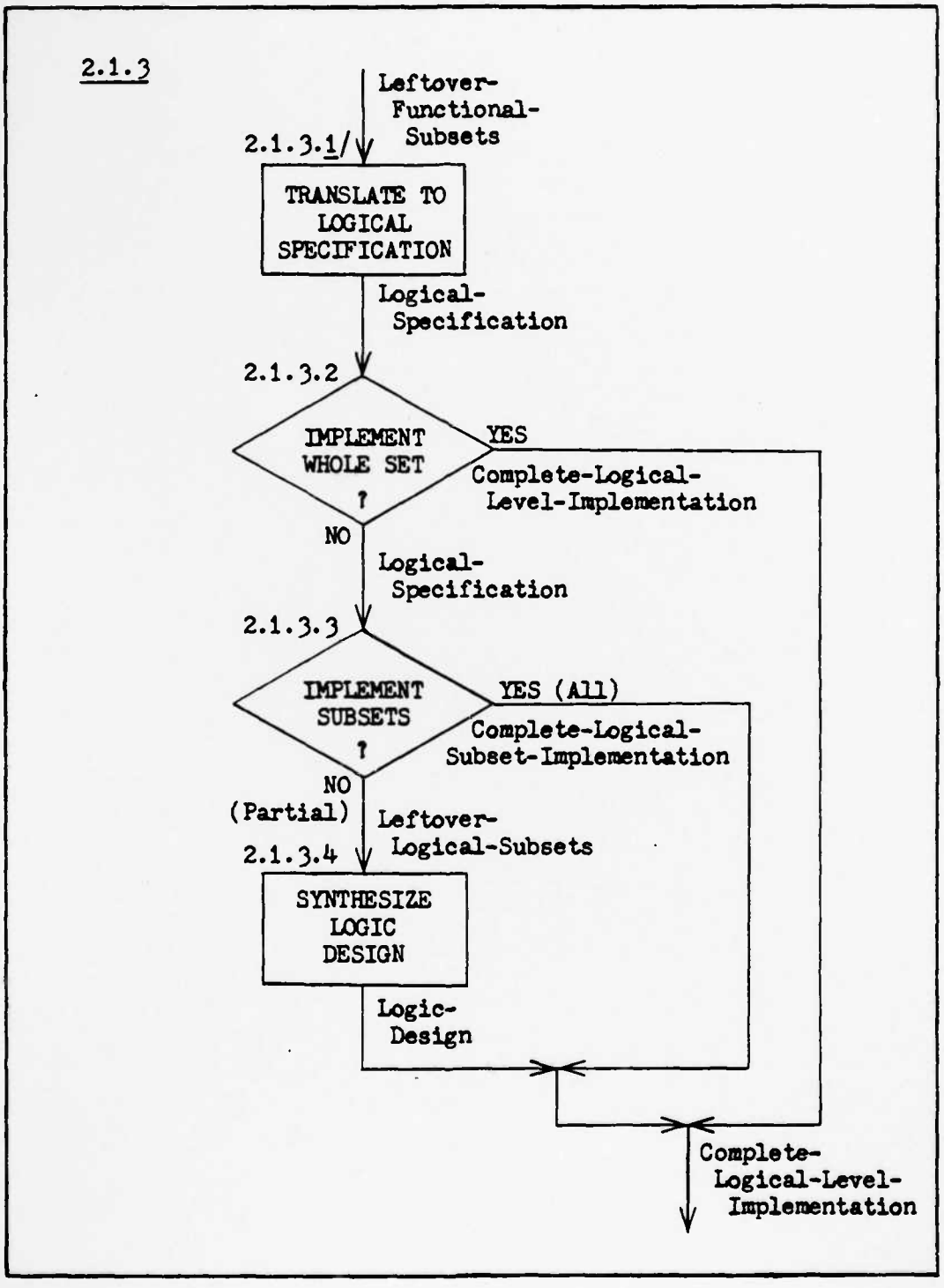
Block 2.1.2


2.1.2.2
Functional-
Specification
LIBRARY
Chip-Families-
&-Functions
.1
EXISTING
FUNCTIONAL
IMPLEMENTATION
?
NO
YES
Complete-
Functional-
Level-
Implementation
Revised-
Functional-
Specification
.4
MODIFY
FUNCTIONAL
SPECIFICATION
Functional-
Specification-
Comparison
.2
EXISTING
IMPLEMENTATION
CLOSE
?
YES
NO
Functional-
Specification
Inter-
action
USER
Close-
Functional-
Implementation
Int.
.3
CLOSE
IMPLEMENTATION
APPLY
?
YES
NO
Close-
Functional-
Implementation
Functional-
Specification

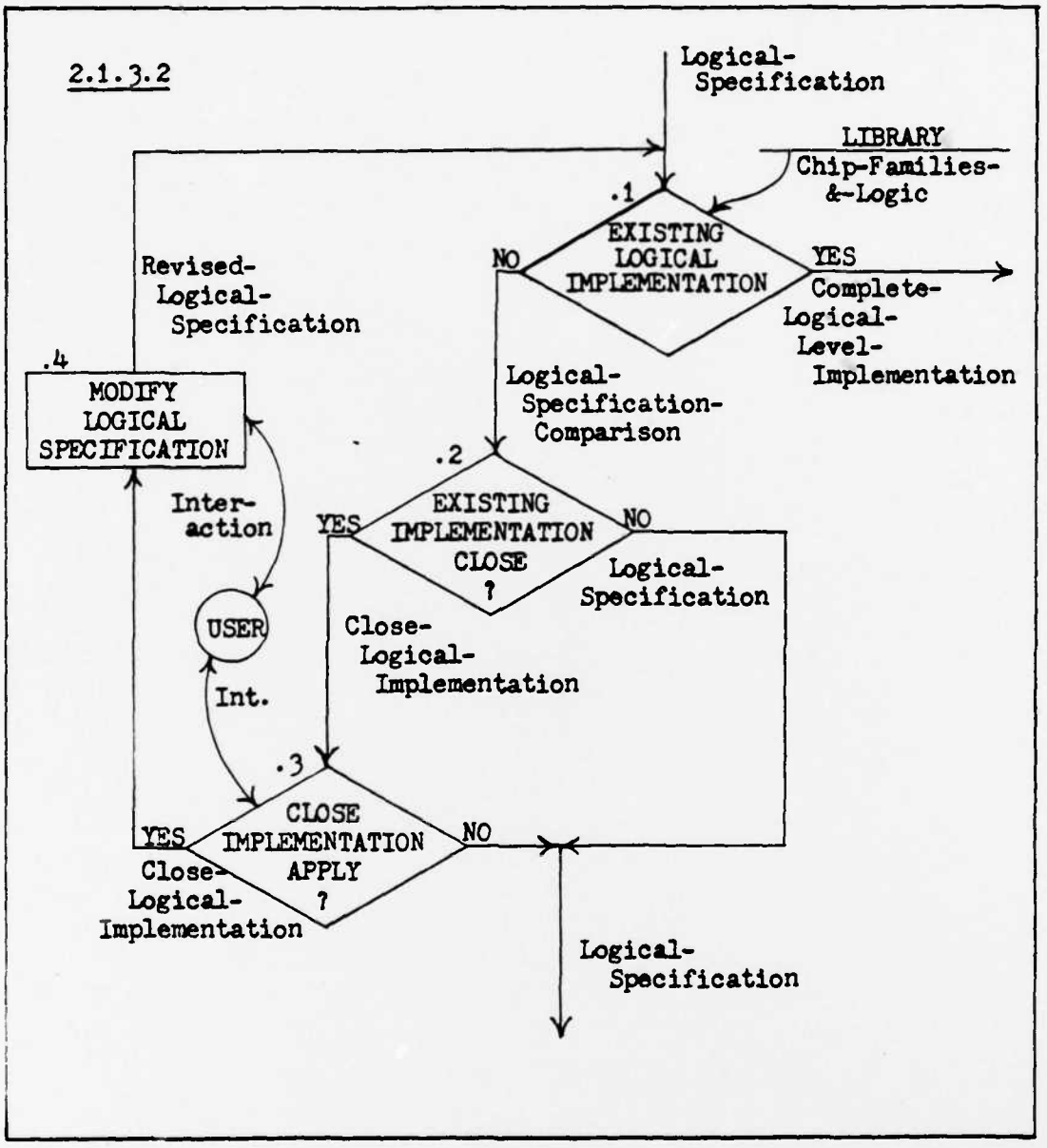
Block 2.1.2.2


2.1.2.3
Functional-Specification
.1
FOR i←1, n ASSIGN SUBSETS
Indexed-Functional-Subsets
.2
i←1
Indexed-Functional-Subsets
.3
SELECT SUBSET i
Chip-Families-&-Functions
LIBRARY
Functional-Subset-i
.4
EXISTING IMPLEMENTATION FOR SUBSET i ?
Subset-i-Comparison
NO
Subset-i-Implementation
YES
.6
EXISTING IMPLEMENTATION CLOSE ?
Close-Subset-i
YES
Subset-i
NO
to "NO" side only
.8
MODIFY SPEC. FOR CLOSE IMPLEM. ?
Unmodified-Subset-i
YES
Unmodified-Subset-i
NO
.15
TRY AGAIN ?
NO
YES
.9
MODIFY SUBSET i SPECIFICATION
Int.
USER
.5
SAVE SUBSET i IMPLEMENTATION
.14
ALLOW IMPACT ?
YES
NO
Int.
USER
Modified-Subset-i
.10
MAP TO IMPLEMENTATION
.7
SAVE SUBSET i
.13
NOTIFY USER
Modified-Subset-i-Implementation
.11
IMPACTS OTHER IMPLEM. SUBSETS ?
YES
Modified-Subset-i-Implementation
NO
.12
SAVE MODIFIED SUBSET i IMPLEMENTATION
.16
IDENTIFY SUBSETS IMPACTED
.18
i←i + 1
.17
i = n ?
YES
NO
.19
LEFTOVER SUBSETS ?
YES
NO
Leftover-Functional-Subsets
Complete-Functional-Subset-Implementation

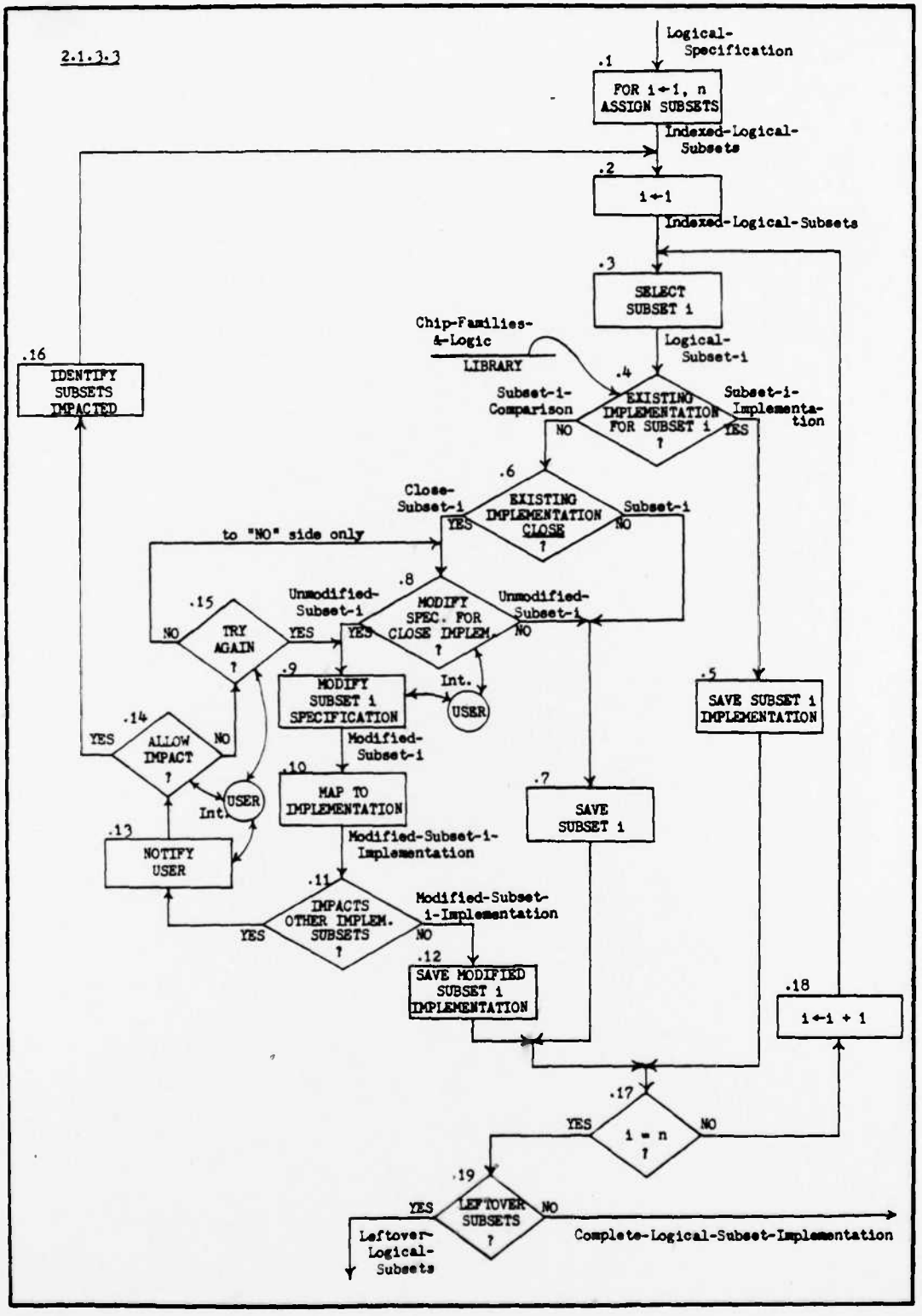
Block 2.1.2.3


2.1.3
Leftover-
Functional-
Subsets
2.1.3.1
TRANSLATE TO
LOGICAL
SPECIFICATION
Logical-
Specification
2.1.3.2
IMPLEMENT
WHOLE SET
?
YES
Complete-Logical-
Level-Implementation
NO
Logical-
Specification
2.1.3.3
IMPLEMENT
SUBSETS
?
YES (All)
Complete-Logical-
Subset-Implementation
NO
(Partial)
Leftover-
Logical-Subsets
2.1.3.4
SYNTHESIZE
LOGIC
DESIGN
Logic-
Design
Complete-
Logical-Level-
Implementation

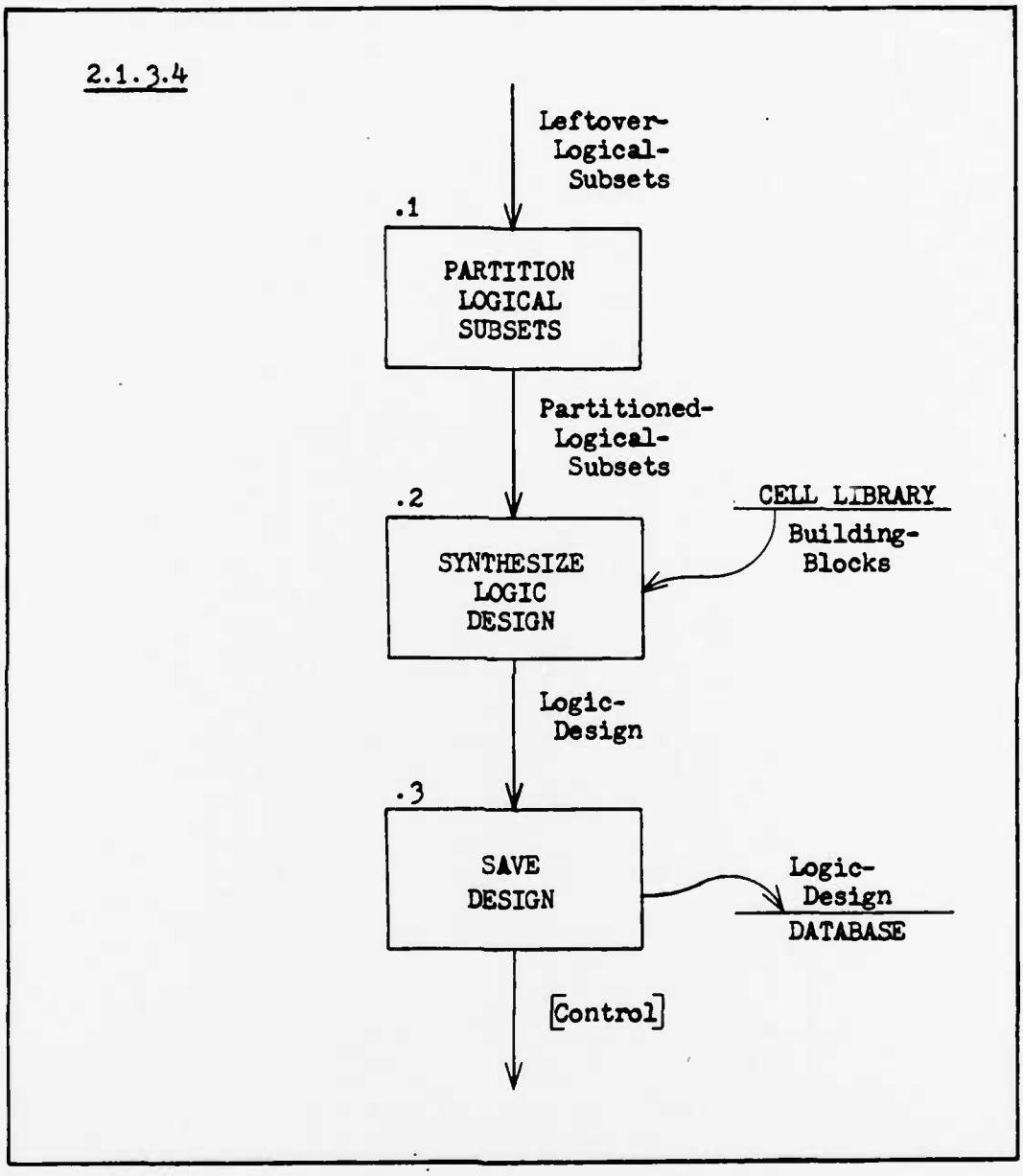
Block 2.1.3


2.1.3.2
Logical-
Specification
LIBRARY
Chip-Families-
&-Logic
.1
EXISTING
LOGICAL
IMPLEMENTATION
NO
YES
Complete-
Logical-
Level-
Implementation
Revised-
Logical-
Specification
.4
MODIFY
LOGICAL
SPECIFICATION
Logical-
Specification-
Comparison
.2
EXISTING
IMPLEMENTATION
CLOSE
?
YES
NO
Logical-
Specification
Inter-
action
USER
Close-
Logical-
Implementation
Int.
.3
CLOSE
IMPLEMENTATION
APPLY
?
YES
NO
Close-
Logical-
Implementation
Logical-
Specification

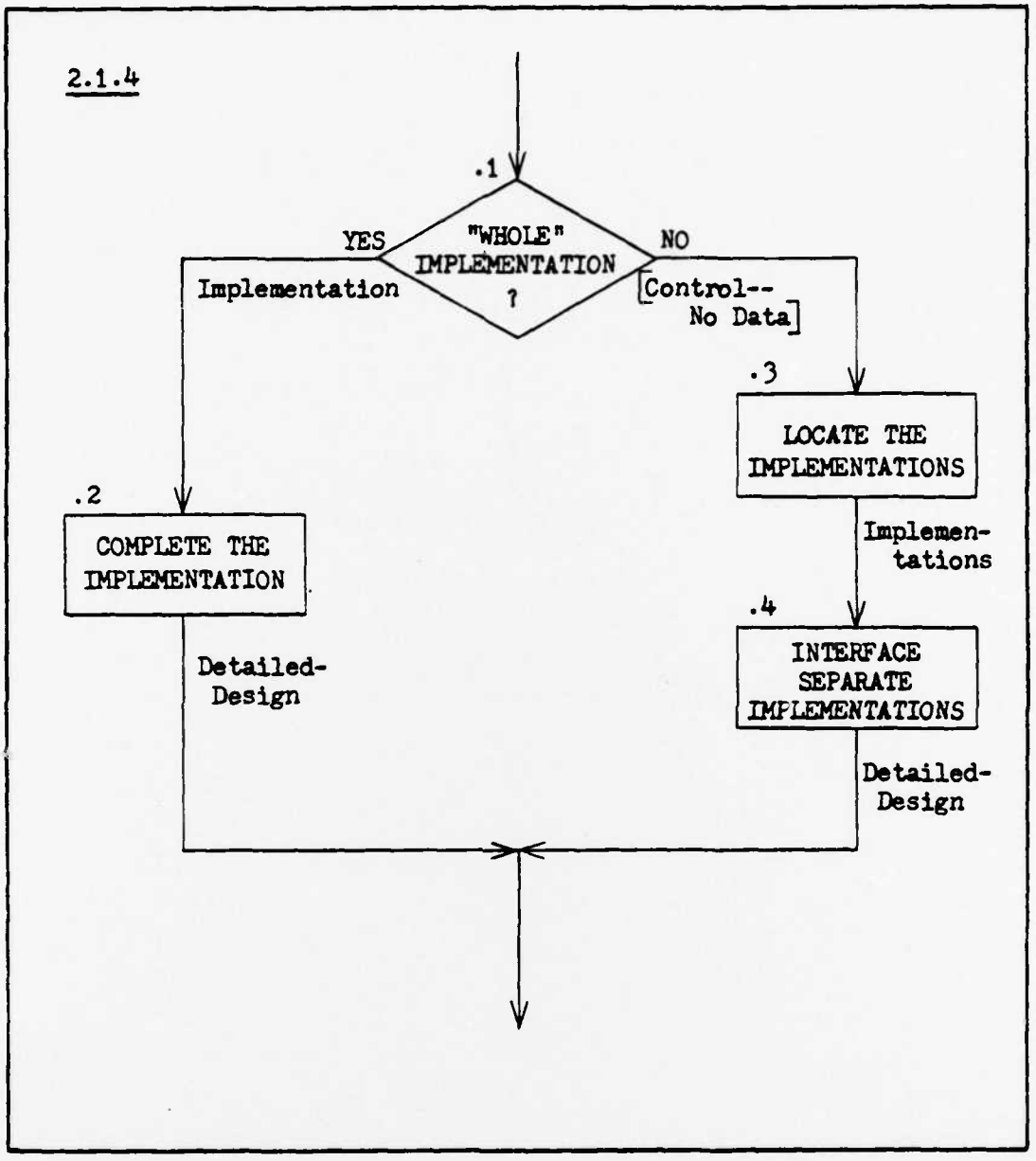
Block 2.1.3.2


2.1.3.3
Logical-Specification
.1
FOR i←1, n ASSIGN SUBSETS
Indexed-Logical-Subsets
.2
i←1
Indexed-Logical-Subsets
.3
SELECT SUBSET i
Chip-Families-&-Logic LIBRARY
Logical-Subset-i
.4
EXISTING IMPLEMENTATION FOR SUBSET i ?
Subset-i-Comparison
Subset-i-Implementation
NO
YES
.6
EXISTING IMPLEMENTATION CLOSE ?
Close-Subset-i
Subset-i
YES
NO
to "NO" side only
.8
MODIFY SPEC. FOR CLOSE IMPLEM. ?
Unmodified-Subset-i
Unmodified-Subset-i
YES
NO
.15
TRY AGAIN ?
NO
YES
.9
MODIFY SUBSET i SPECIFICATION
Int.
USER
Modified-Subset-i
.14
ALLOW IMPACT ?
YES
NO
USER
Int.
.10
MAP TO IMPLEMENTATION
Modified-Subset-i-Implementation
.13
NOTIFY USER
.11
IMPACTS OTHER IMPLEM. SUBSETS ?
YES
NO
Modified-Subset-i-Implementation
.12
SAVE MODIFIED SUBSET i IMPLEMENTATION
.16
IDENTIFY SUBSETS IMPACTED
.5
SAVE SUBSET i IMPLEMENTATION
.7
SAVE SUBSET i
.18
i←i + 1
.17
i = n ?
YES
NO
.19
LEFTOVER SUBSETS ?
YES
NO
Leftover-Logical-Subsets
Complete-Logical-Subset-Implementation

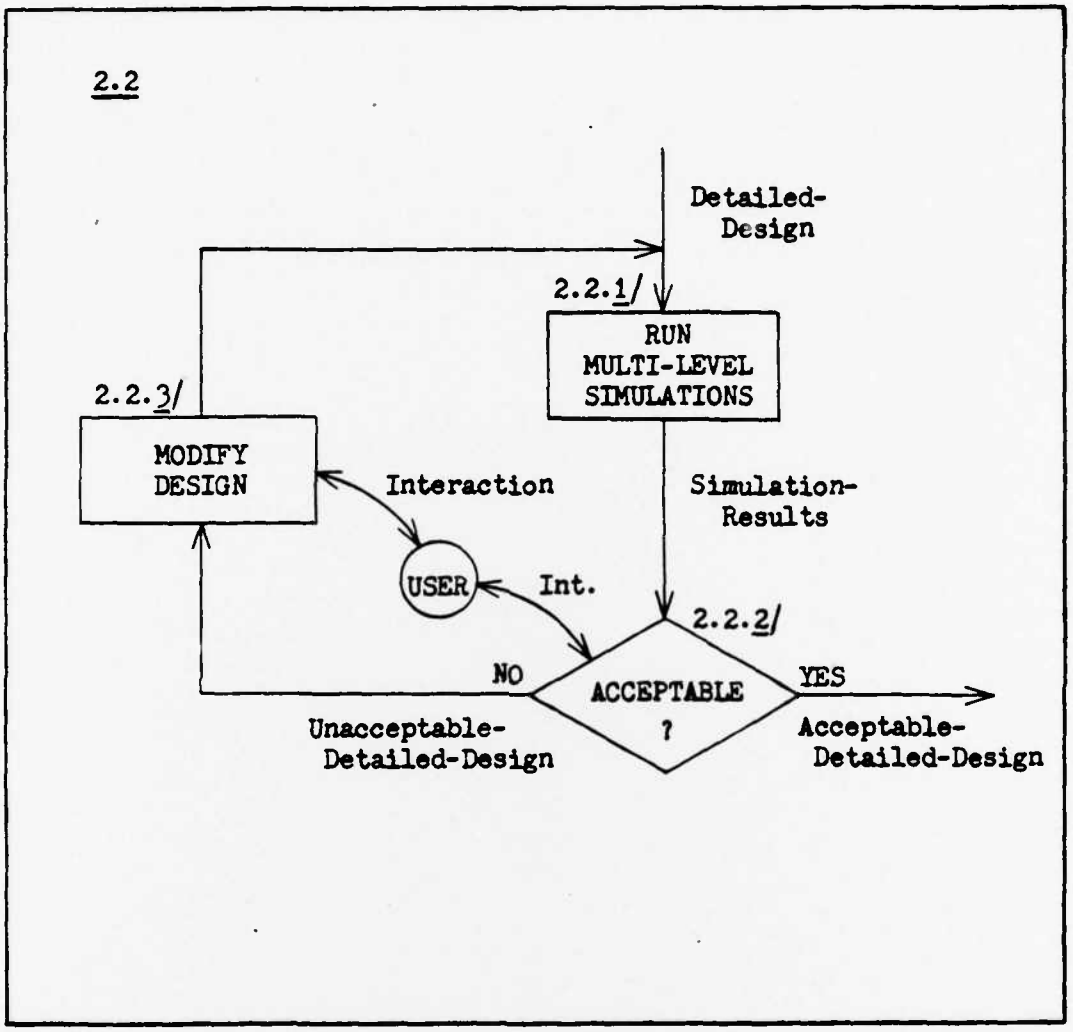
Block 2.1.3.3


2.1.3.4
Leftover-
Logical-
Subsets
.1
PARTITION
LOGICAL
SUBSETS
Partitioned-
Logical-
Subsets
.2
CELL LIBRARY
Building-
Blocks
SYNTHESIZE
LOGIC
DESIGN
Logic-
Design
.3
SAVE
DESIGN
Logic-
Design
DATABASE
[Control]

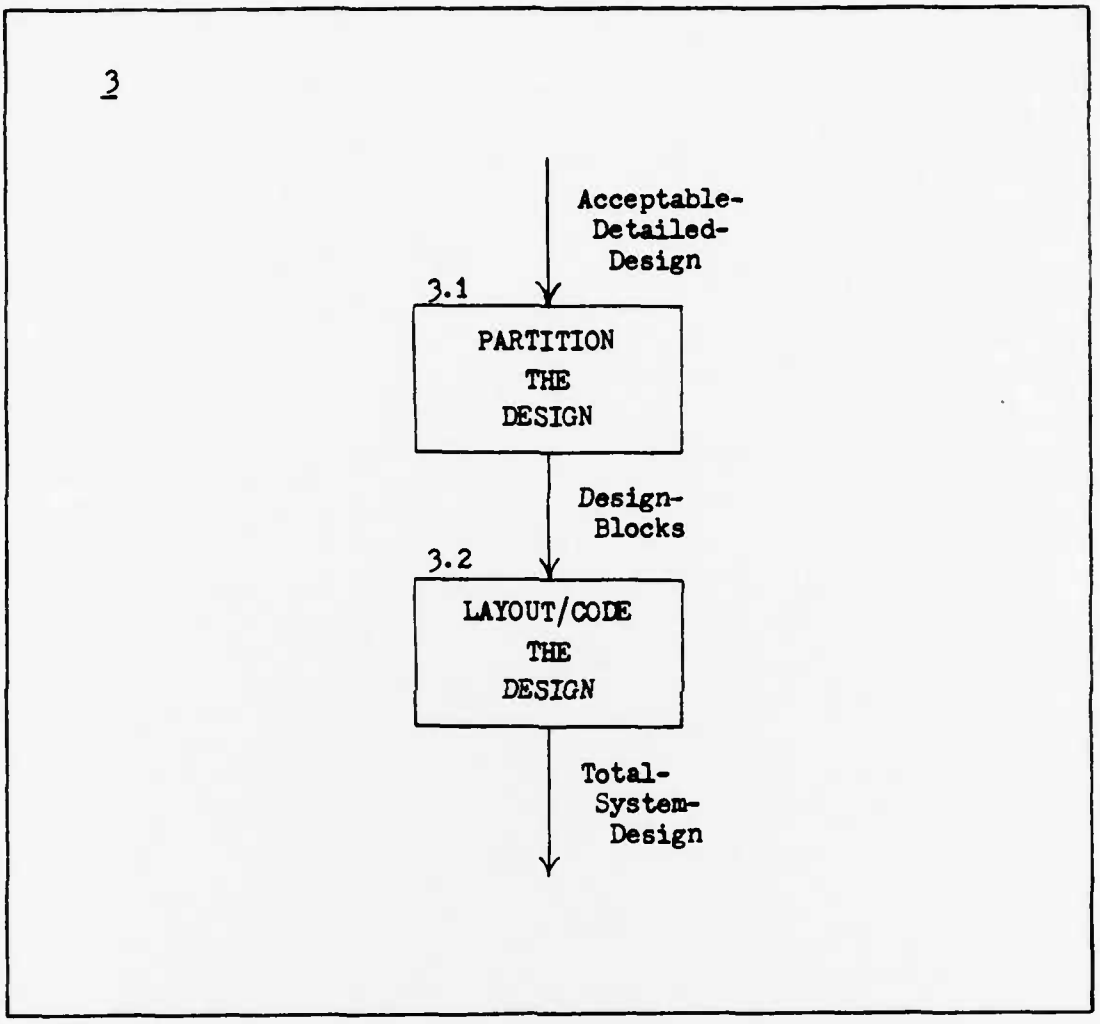
Block 2.1.3.4


2.1.4
.1
"WHOLE" IMPLEMENTATION ?
YES
NO
Implementation
[Control-- No Data]
.3
LOCATE THE IMPLEMENTATIONS
.2
COMPLETE THE IMPLEMENTATION
Implemen-tations
.4
INTERFACE SEPARATE IMPLEMENTATIONS
Detailed-Design
Detailed-Design

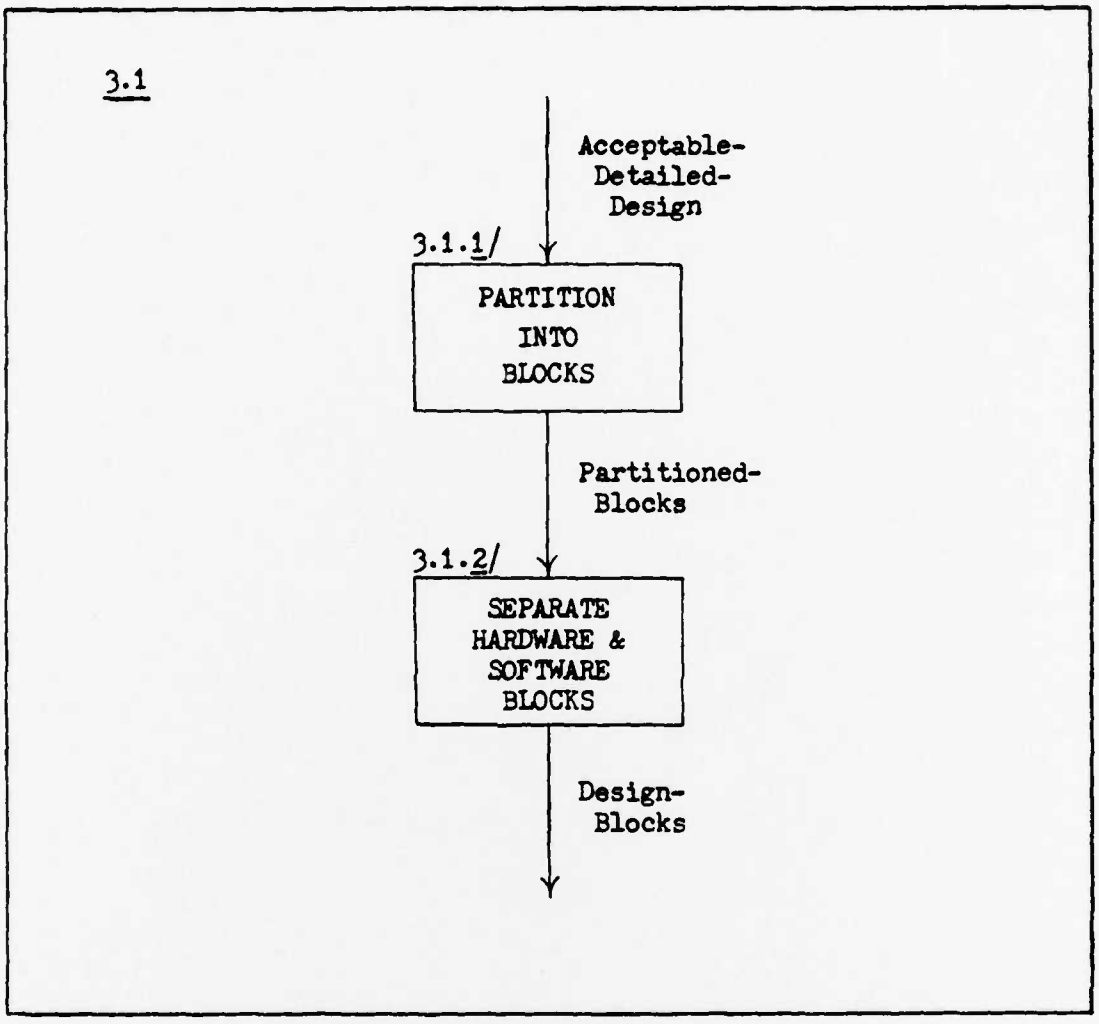
Block 2.1.4


2.2
Detailed-
Design
2.2.1/
RUN
MULTI-LEVEL
SIMULATIONS
2.2.3/
MODIFY
DESIGN
Interaction
Simulation-
Results
USER
Int.
2.2.2/
NO
ACCEPTABLE
?
YES
Unacceptable-
Detailed-Design
Acceptable-
Detailed-Design

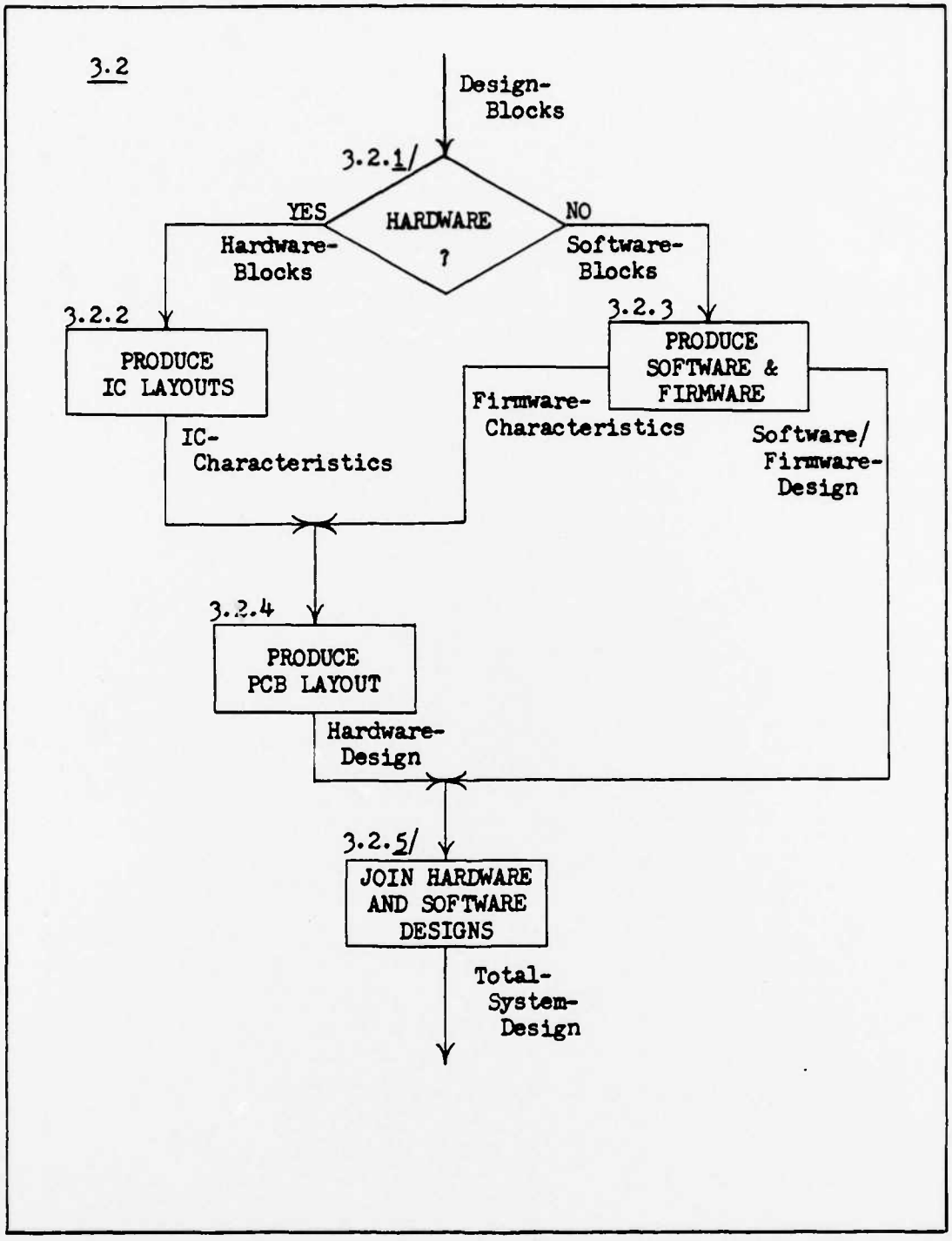
Block 2.2


3
Acceptable-
Detailed-
Design
3.1
PARTITION
THE
DESIGN
Design-
Blocks
3.2
LAYOUT/CODE
THE
DESIGN
Total-
System-
Design

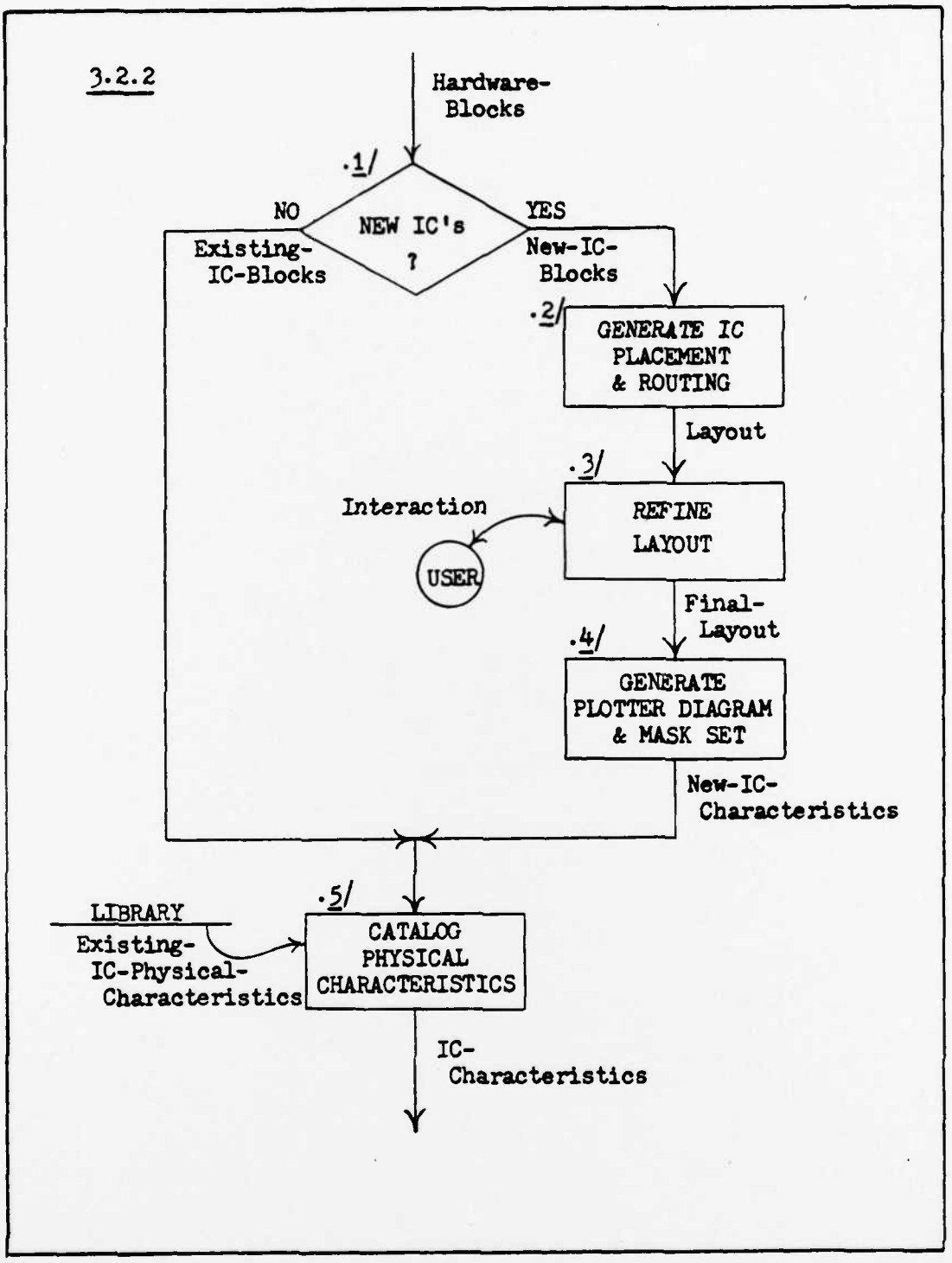
Block 3


3.1
Acceptable-
Detailed-
Design
3.1.1/
PARTITION
INTO
BLOCKS
Partitioned-
Blocks
3.1.2/
SEPARATE
HARDWARE &
SOFTWARE
BLOCKS
Design-
Blocks

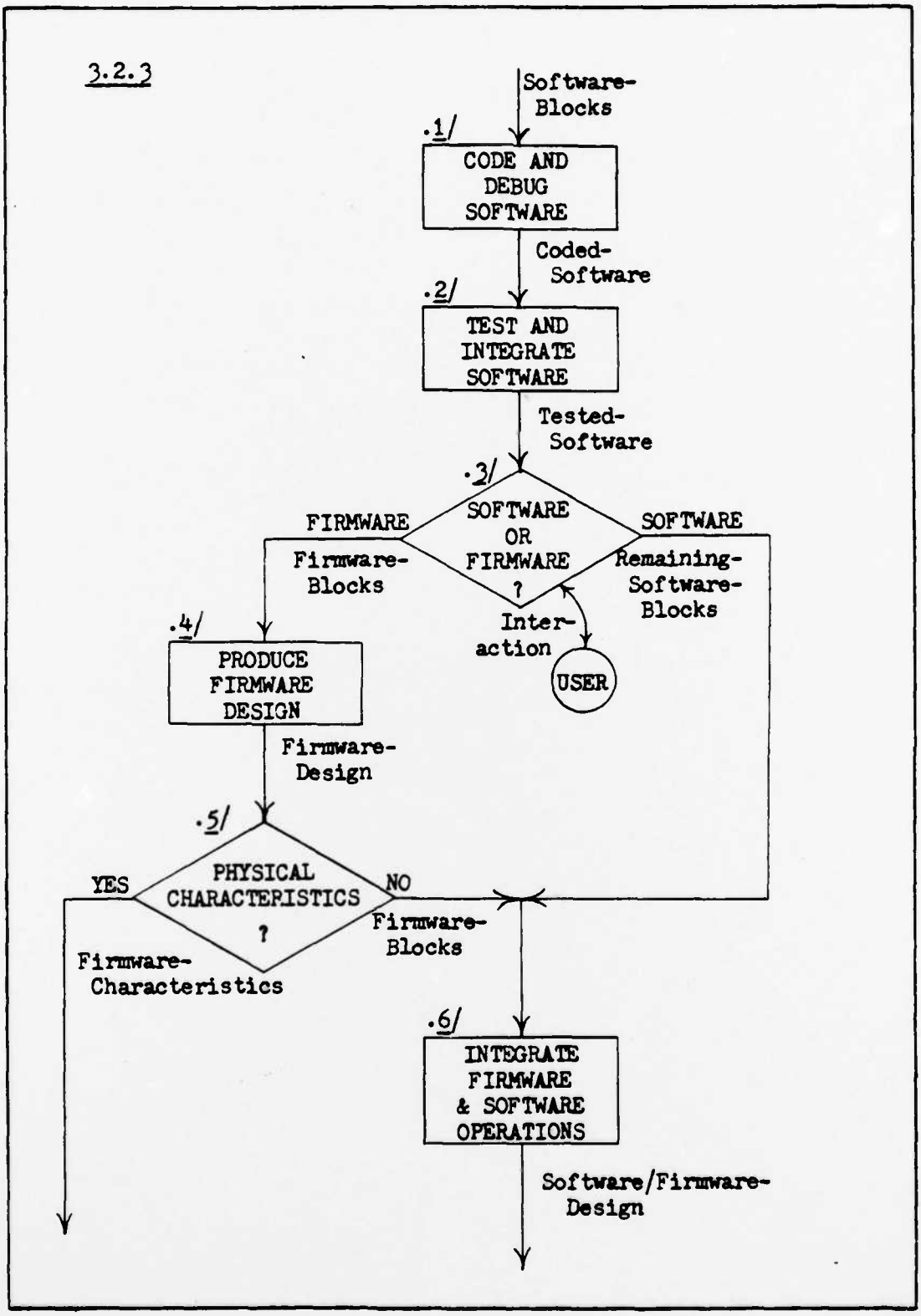
Block 3.1


3.2
Design-
Blocks
3.2.1/
HARDWARE
?
YES
NO
Hardware-
Blocks
Software-
Blocks
3.2.2
PRODUCE
IC LAYOUTS
3.2.3
PRODUCE
SOFTWARE &
FIRMWARE
IC-
Characteristics
Firmware-
Characteristics
Software/
Firmware-
Design
3.2.4
PRODUCE
PCB LAYOUT
Hardware-
Design
3.2.5/
JOIN HARDWARE
AND SOFTWARE
DESIGNS
Total-
System-
Design

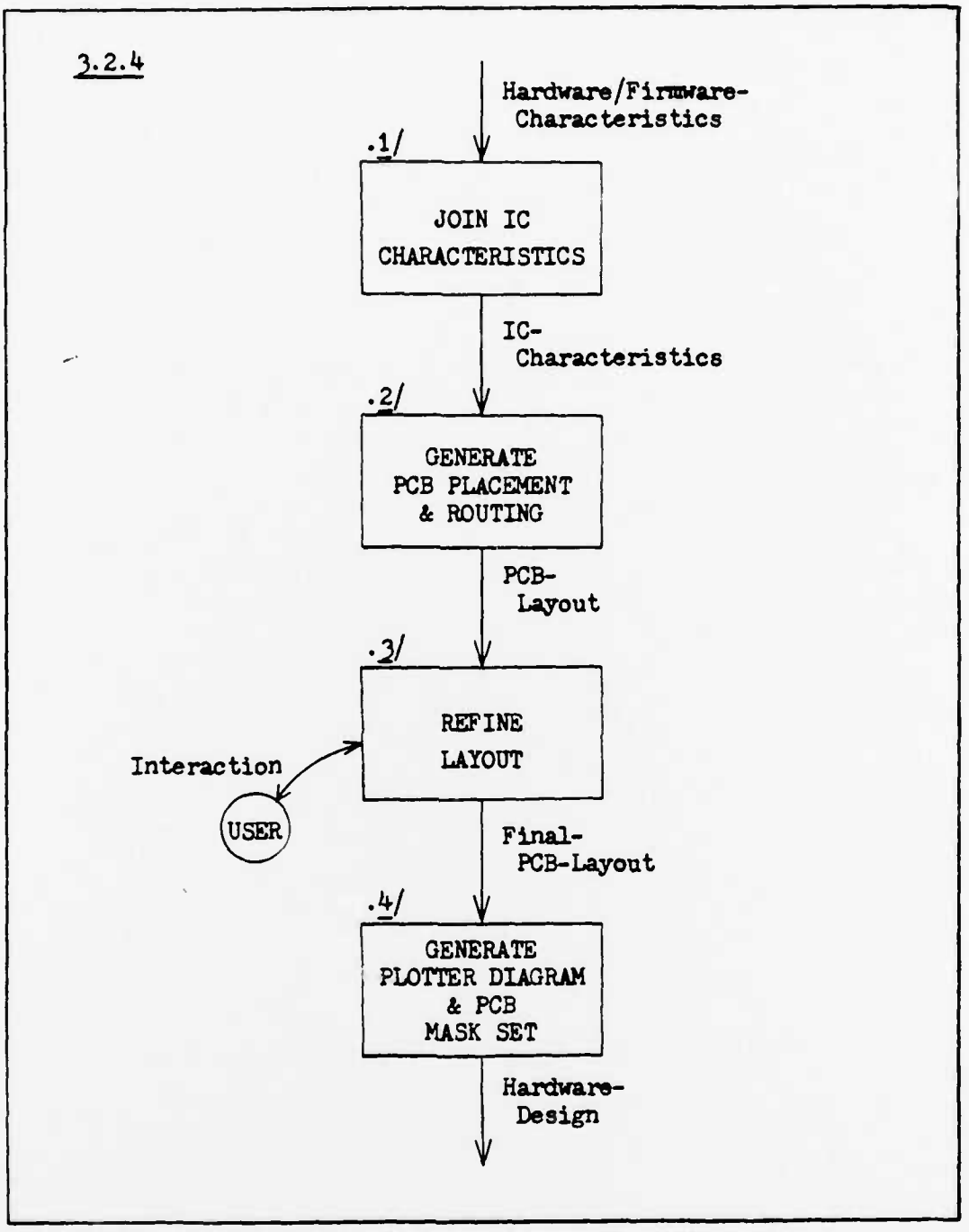
Block 3.2


3.2.2
Hardware-Blocks
.1/
NEW IC's ?
NO
Existing-IC-Blocks
YES
New-IC-Blocks
.2/
GENERATE IC PLACEMENT & ROUTING
Layout
.3/
REFINE LAYOUT
Interaction
USER
Final-Layout
.4/
GENERATE PLOTTER DIAGRAM & MASK SET
New-IC-Characteristics
.5/
LIBRARY
Existing-IC-Physical-Characteristics
CATALOG PHYSICAL CHARACTERISTICS
IC-Characteristics

Block 3.2.2


3.2.3
Software-Blocks
.1/
CODE AND DEBUG SOFTWARE
Coded-Software
.2/
TEST AND INTEGRATE SOFTWARE
Tested-Software
.3/
SOFTWARE OR FIRMWARE ?
FIRMWARE
Firmware-Blocks
SOFTWARE
Remaining-Software-Blocks
Inter-action
USER
.4/
PRODUCE FIRMWARE DESIGN
Firmware-Design
.5/
PHYSICAL CHARACTERISTICS ?
YES
NO
Firmware-Characteristics
Firmware-Blocks
.6/
INTEGRATE FIRMWARE & SOFTWARE OPERATIONS
Software/Firmware-Design

Block 3.2.3


3.2.4
Hardware/Firmware-
Characteristics
.1/
JOIN IC
CHARACTERISTICS
IC-
Characteristics
.2/
GENERATE
PCB PLACEMENT
& ROUTING
PCB-
Layout
.3/
REFINE
LAYOUT
Interaction
USER
Final-
PCB-Layout
.4/
GENERATE
PLOTTER DIAGRAM
& PCB
MASK SET
Hardware-
Design

Block 3.2.4

VITA

Hobart S. Cable, II, was born in Canton, Ohio, on 28 September 1942. After graduation from high school in 1960 he spent a year at Ohio State University and four years at the USAF Academy in Colorado. He graduated in 1965 with a Bachelor of Science degree in Basic Sciences and a commission in the U. S. Air Force. He then attended Pilot Training at Reese AFB, Texas, and received his wings in August 1966. For the next five years he flew weather reconnaissance C-130's investigating hurricanes and typhoons, with time out for a year at Nha Trang Air Base, Republic of Vietnam, flying specially equipped "Black Bird" C-130's. In February 1972 he arrived at Andrews AFB, Maryland, for almost seven years as a pilot with the 89th Military Airlift Wing, Special Missions. During that time he flew the VC-131H, VC-6A, and VC-9C aircraft, piloting many U. S. and foreign dignitaries, including three Vice Presidents, the First Family, and several presidents of other countries. He transferred to Wright-Patterson AFB, Ohio, in October 1978 as a Display Systems Engineer for the Avionics Engineering Directorate of Aeronautical Systems Division. In January 1979 he began the part-time study program at the Air Force Institute of Technology for a Master's Degree in Computer Systems.

Permanent address: 5560 E. Blvd. NW

Canton, Ohio 44718

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER: AFIT/GCS/EE/82J-7 | 2. GOVT ACCESSION NO.: AD-A118039 |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE (and Subtitle): SUPER-CAD: AN INTEGRATED STRUCTURE FOR DESIGN AUTOMATION | 5. TYPE OF REPORT & PERIOD COVERED: MS Thesis |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): Hobart S. Cable, II, Major, USAF | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS | 12. REPORT DATE: June 1982 |
| | 13. NUMBER OF PAGES: 139 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | 15. SECURITY CLASS. (of this report): Unclassified |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17 23 JUL 1982

LYNN E. WOLAVER
Dean for Research and Professional Development

AIR FORCE INSTITUTE OF TECHNOLOGY (ATC)
WRIGHT-PATTERSON AFB, OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

| | | |
|---|---|---|
| Computer Aided Design | IC Design | VHSIC |
| Design Automation | Logic Synthesis | VLSI |
| Digital Electronics Design | Placement and Routing | |
| Digital Systems Design | Simulation | |
| Integrated Circuit Design | | |

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This project proposes a structure to integrate a variety of Computer-Aided Design tools into a complete design system. Design aids have provided valuable assistance to designers of integrated circuits over the last decade. However, greatly increased circuit complexity, with the approach of more than a million devices on a single chip, is exceeding the capabilities of current design methods. Greater automation and additional design tools are needed.

A model is proposed which divides the design process into three stages:

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Specification, Implementation, and Realization. The Implementation stage is examined in detail. Design requirements can be described at different levels of abstraction, from a description of overall behavior to specific gate-level logic details. The model attempts to satisfy the requirements at the higher levels using existing implementations before resorting to the design of new circuits at the lowest level.

The proposed system is an integral part of the Air Force Institute of Technology's increased emphasis on Design Automation. As future efforts address more of the details and problem areas, design tools will be developed to support it. The system will be highly flexible, based on a great degree of interaction with the user, and adaptable to changing technologies and requirements.


FILMED
9 8